Universidad Carlos III de Madrid
Escuela Politécnica Superior de Leganés
Grado en Ingeniería Electrónica Industrial y Automática

Trabajo Fin de Grado

UAV Flight controller using UDOO

Jorge Plaza Buquerín

Tutor: Abdulla Hussein Abdulrahman Al Kaff

Departamento de Ingeniería de Sistemas y Automática

Fecha de entrega	Fecha de defensa
26 de Septiembre de 2016	4 de Octubre de 2016

Agradecimientos

En primer lugar, quiero agradecer a mi familia el apoyo que me han dado durante la realización de este proyecto. Sus preguntas acerca del proyecto no han hecho más que incentivarme y animarme a mejorar el trabajo realizado. También me gustaría dar las gracias a la Universidad Carlos III de Madrid por poner los medios para la realización del mismo. En particular, me gustaría agradecer al departamento de sistemas inteligentes y a todos sus miembros por haberme acogido en esa gran familia y por haberme soportado, teniendo en cuenta que mi proyecto era el más ruidoso del laboratorio. Con respecto a mi tutor, el profesor Abdulla Hussein Al Kaff, sólo tengo palabras de agradecimiento por empujarme a mejorar día a día y por su exhaustiva supervisión a lo largo de todo el proyecto. Por otro lado, me gustaría dar las gracias a Ahmed Radwan Ibrahim por haber colaborado en el proyecto demostrando un entusiasmo, unos conocimientos y una paciencia admirables. No solo es un ingeniero excepcional, sino también un gran amigo. Por último, me gustaría agradecer a todos mis compañeros y amigos de la Universidad, en especial a Manuel Peña Fernández, Carlos Romero Blanco y Jorge Rodríguez de Frutos por su interés acerca del estado del proyecto.

Resumen

Hoy en día, el sector de los vehículos aéreos no tripulados está creciendo rápidamente ya que existen multitud de aplicaciones que pueden ser potencialmente sustituidas por el uso de UAV. Aunque existen aplicaciones en las que es más adecuado utilizar UAV de ala fija, en la mayoría de los casos se utilizan multirotores debido a que son sistemas capaces de realizar un despegue y aterrizaje vertical (VTOL), lo que les hace idóneos para vuelos estacionarios. Los multirotores tienen multitud de ventajas, sin embargo, desde el punto de vista del control son sistemas sub-actuados, por lo que precisan de un sistema de control para estabilizarse. En el presente proyecto, se ha desarrollado el sistema de control vuelo de un cuadricóptero controlado remotamente, basado en la regulación PID. El sistema propuesto se ha implementado en la placa UDOO, que contiene un microcontrolador ARM A9 y un Arduino Due. Además se han realizado pruebas de simulación gracias al modelado en Matlab. Los resultados obtenidos tras los ensayos realizados en el banco de pruebas, muestran que el sistema de control de estabilización diseñado es capaz de mantenerse en torno al punto deseado con un error bastante pequeño. En definitiva, el sistema diseñado cumple con los objetivos de crear un UAV de bajo coste, con un sistema de control adecuado y que nos sirva como base para crear más adelante un UAV autónomo.

Abstract

Nowadays, the unmanned aerial vehicle sector is quickly growing because there are a lot of applications that can be potentially substituted for the use of UAV. Although there are applications in which is more appropiate to use fixed-wing UAV, in most cases multirotors are prefered because they are systems able to do vertical take-off and landing (VTOL), so they are suitable for stationary flights. Multirotors have a lot of advantages, however from the point of view of control they are sub-actuated systems, so that they need a control system to stabilize themselves. In this proyect, a flight control system is developed for a remotely controlled quadcopter, based on a PID controller. The system proposed is implemented on the UDOO board, which contains an ARM A9 microcontroller and an Arduino Due compatible. Also, we have done simulation tests based on the quadcopter model implemented in Matlab. The results obtained after the experiments done in the test bench, shows that the stabilization control system is able to track the setpoint with a small error. In conclusion, the designed system meets the objetives of creating a low-cost UAV with an adecuated control system that can be used in the future as base of an autonomous quadcopter.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1 Introducción

1.1 Contexto

Durante los últimos años, se ha producido un gran auge en el desarrollo de los UAV (*Unmanned Aerial Vehicle*), impulsado tanto por el avance de la electrónica como por la necesidad de buscar soluciones a ciertas aplicaciones que conllevan un riesgo notable para el ser humano o un gran coste. Los vehículos aéreos no tripulados, como muchas otras tecnologías que utilizamos actualmente, nacieron en el siglo XX con el fin de utilizarlos en aplicaciones militares. Sin embargo, con el paso del tiempo, la industria civil se dio cuenta de que los UAV no sólo se podían utilizar con fines bélicos, sino que se podrían utilizar como herramienta para multitud de aplicaciones.

Dentro de la industria aeroespacial, los sistemas aéreos no tripulados son el sector en el que se está produciendo el mayor crecimiento. El estudio de mercado "*Worldwide UAV Production Will Total $93 Billion in Its 2015 UAV Market Profile and Forecast*" realizado en 2015 por Teal Group estima que en los próximos diez años los UAV van a pasar de producir los 4.000 millones de dólares actuales hasta alcanzar los 93.000 millones de dólares. Actualmente, el 72% del mercado de UAV corresponde al campo militar mientras el 28% restante corresponde a la industria civil y al consumo. Pero es en el campo civil donde se va a producir un mayor crecimiento, lo cual hace que otras compañías fuera del sector como Amazon o Google se estén interesando cada vez más.

Hoy en día, el marco legislativo que rodea los UAV, o como se les denomina actualmente RPAS (*Remotely Piloted Aircraft Systems*), limita su uso en aplicaciones civiles y obliga a realizar las mismas con unos operadores certificados por la unión europea. Razón por la cual se está trabajando en una legislación europea más permisiva que saldrá a la luz en los próximos dos años y que permitirá que el sector desarrolle al máximo su potencial. En este sentido, el estudio *"Clarity from above. PwC global report on the comercial applications of drone technology"* realizado por Price Waterhouse & Coopers en Mayo de 2016, ha analizado los servicios con alto potencial de ser sustituidos por RPAS y el valor de mercado de dichos servicios.

Los sectores ordenados de mayor a menor potencial son: Infraestructuras gestión, mantenimiento e inventario), agricultura (supervisión y evaluación de cultivos), transporte (paquetería y suministro de repuestos), seguridad (vigilancia y monitorización), entretenimiento y medios de comunicación (anuncios, fotografía aérea), seguros (detección de fraudes), telecomunicaciones (mantenimiento e inspección de torres) y minería (exploración e impacto medioambiental).

Según este estudio, el valor total de los servicios actuales de negocios y mano de obra que pueden ser reemplazados por el uso de UAV en un futuro próximo está valorado en 127.3 miles de millones de dólares.

También se podrían utilizar UAV en otras aplicaciones civiles como la prevención de incendios, la búsqueda y rescate de personas, la cartografía, la inspección de líneas eléctricas, el control del medio ambiente, aplicaciones orientadas a la docencia, etc.

Estos son sólo unos pocos ejemplos de las aplicaciones de estos sistemas, lo cual explica el crecimiento que se está produciendo. Por otro lado, el desarrollo de la electrónica nos brinda la posibilidad de utilizar microcontroladores cada vez más potentes, cuyas dimensiones y peso nos permiten integrarlo a bordo del UAV. Estos microcontroladores actúan como ordenadores de abordo y nos permiten conectar redes de sensores y realizar al mismo tiempo operaciones costosas como, por ejemplo, el procesamiento de imágenes.

Tanto los UAV autónomos como los controlados remotamente, reciben una gran cantidad de información del exterior a través de diversos sensores como sistemas inerciales (IMU), sensores barométricos para medir la altura, LIDAR para detectar distancias y realizar mapeos del entorno, GPS para la localización, cámaras, sensores de ultrasonidos para detectar obstáculo, etc. Por tanto, es necesario que el microcontrolador posea una gran capacidad de procesamiento para juntar toda esa información, lo que se conoce como fusión sensorial.

1.2 Motivación y objetivos del proyecto

Este proyecto nace de la voluntad de crear un vehículo aéreo VTOL (*Vertical Take-Off and Landing*), que nos permita realizar tanto vuelo pilotado mediante radio control como vuelo autónomo con el menor coste posible. Hoy en día, los UAV comerciales tienen un grado de autonomía bajo, son sistemas muy cerrados y tienen un precio bastante alto, por lo que se propone crear el sistema desde cero.

Por tanto, el objetivo del proyecto será diseñar y construir un UAV low-cost, más en concreto un multicóptero que imprimiremos en 3D, y realizar el sistema de control de vuelo utilizando la placa UDOO Quad. De este modo, trataremos de realizar nuestro propio autopiloto que realice la estabilización del sistema, mediante el uso de reguladores PID. También será necesario estimar el estado del cuadricóptero con la información recibida de la IMU y realizar la lectura de los datos enviado por radio control lo más rápido y preciso posible. Para facilitar el ajuste del PID, se propone implementar un modelo del quadcopter en Matlab. Finalmente, se han de realizar ensayos en un banco de pruebas antes de realizar las pruebas de vuelo reales para no comprometer la seguridad del UAV y de las personas que lo rodean.

Por último, cabe destacar que se pretende diseñar un sistema sencillo de modificar y totalmente accesible para realizar después un control a más alto nivel. En este sentido, se ha realizado una memoria lo más explicativa posible puesto que otros investigadores continuarán este proyecto hasta convertirlo en un sistema totalmente autónomo, por lo que no es un proyecto aislado, sino que pertenece a un proyecto de investigación más completo.

1.3 Estructura de la memoria

Se ha estructurado la memoria de manera que en cada capítulo se ha desarrollado una parte del sistema teniendo en cuenta los objetivos marcados.

- Capítulo 1: Se realiza una breve introducción en la que se aborda la importancia del sector de los UAV, la motivación y los objetivos del proyecto.
- Capítulo 2: En el estado del arte se explican los tipos de UAV en general y los cuadricópteros en particular, así como sus fundamentos de vuelo, los métodos de estimación de la actitud y algunas de las estrategias de control de la literatura.
- Capítulo 3: En éste capítulo se realiza la selección de los elementos que componen el quadcopter, algunos de los cuales son dependientes de los otros.
- Capítulo 4: En éste capítulo se realiza la lectura de los datos del receptor RC y de la unidad de medición inercial. Además se realiza el driver de los variadores electrónicos que controlan los motores. Por último se explica el conexionado de los elementos y se diseña una PCB para facilitar dicha labor.
- Capítulo 5: Se explican el sistema de control diseñado el cual está basado en la regulación PID y se muestra un flujograma que describe el funcionamiento del sistema completo.
- Capítulo 6: Se define el modelado matemático del cuadricóptero y se realizan simulaciones del sistema de control para analizar su respuesta.
- Capítulo 7: Se realizan distintos ensayos de vuelo en un entorno de trabajo controlado para ajustar las ganancias del sistema de control.
- Capítulo 8: En las conclusiones se analiza el trabajo realizado y la consecución de los objetivos. Además se exponen las principales líneas de actuación de los trabajos futuros.
- Anexos: En este apartado se expone el diagrama de tiempos del proyecto y el presupuesto del trabajo realizado.

Capítulo 2 Estado del arte

2.1 Introducción a los UAV

Un UAV es una aeronave que vuela sin un piloto a bordo. Es posible diferenciarlos en función del método de control de vuelo que utilicemos:

- **Controlado por radio control:** Se pilota el UAV mediante control remoto a través de un transmisor RC.
- **Autónomo:** El UAV se guiará por sus propios sistemas y gracias a la información que recibe de los sensores que lleva integrados.
- **Preprogramado:** Únicamente será necesario diseñar el plan de vuelo, este tipo de control se utiliza habitualmente para aplicaciones de vigilancia.
- **Monitorizado:** Es necesaria la intervención de un técnico humano para proporcionar información al UAV con el fin de decidir qué acción se llevará a cabo. Es decir, tiene un plan de vuelo preprogramado pero es posible interactuar con el dron para decidir las tareas que se han de realizar. Se utilizan con fines militares.

Existen gran variedad de formas de clasificar los vehículos aéreos no tripulados, pero principalmente se pueden dividir en dos grandes bloques en función de la finalidad de sus aplicaciones:

- **UAV militares:** Comúnmente son llamados UCAV, por sus siglas en Inglés *Unmanned Combat Air Vehicle*, se diseñan para el ejército y suelen llevar capacidad para explosivos. Este tipo de drones se utilizan para tareas de reconocimiento, vigilancia de fronteras y, como su propio nombre indica, para el combate.
- **UAV de uso civil:** Actualmente, los UAV sólo representan el 28% del total de la industria, sin embargo durante los últimos años está aumentando considerablemente su uso en diversas aplicaciones. Podemos distinguirlos para usos comerciales (dar apoyo a empresas y autónomos para la realización de videos, fotografías, cartografía, transporte, etc.), para aficionados (últimamente muy solicitados por los amantes de la tecnología) y drones utilizados por el gobierno (utilizados en labores de salvamento y rescate, prevención de incendios, vigilancia y control de zonas peligrosas).

Una vez conocidos los tipos de UAV en general y el uso específico de cada uno de ellos, nos centraremos en los drones de uso civil que existen según su morfología:

- **UAV de ala fija:** Es una aeronave compuesta por un ala rígida que tiene una superficie aerodinámica capaz de sustentarla y un motor eléctrico o de combustión interna que hace girar la hélice que proporciona el empuje necesario para volar. Es capaz de realizar largos vuelos, pero necesita estar en movimiento para sustentarse y no puede permanecer en una posición fija. Además necesita una gran superficie para realizar el despegue y el aterrizaje.

- **UAV de ala rotatoria o multirotor**: Consiste en dos o tres palas de rotor que giran en torno a una estructura fija, lo que se conoce como rotor. Existen multitud de configuraciones en función del número de rotores que contenga (Helicopter, Tricopter, Quadcopter, Hexacopter, etc.). Son UAV más complejos puesto que el modelo que representa el comportamiento aerodinámico del multirotor es no lineal. La ventaja principal de los multirotor es la capacidad para realizar el despegue y aterrizaje vertical (VTOL), lo cual le hace idóneo para aplicaciones estacionarias como trabajos de inspección.

2.1.1 Multicópteros

Un multicóptero es una aeronave cuyo movimiento se controla mediante la aceleración y deceleración de los distintos motores que lo componen. Dichos motores mueven las hélices que proporcionan el empuje necesario para sustentarlo. Dependiendo del tipo de configuración, los movimientos de los motores se realizarán de una forma u otra. A continuación describiremos las principales configuraciones utilizadas en multicópteros.

El Tricopter o tricóptero está compuesto por una estructura con tres ejes situados formando ángulos de 120° entre sí, y un motor en cada uno de ellos. Los tricópteros requieren que uno de los motores esté guiado por un servo, el cual inclina el motor con el fin de poder realizar todos los movimientos.

Figura 2.1 Ejemplo Tricopter [21]

El Quadcopter es el tipo de multicóptero más utilizado hoy en día para aplicaciones civiles, puesto que tiene una estructura sencilla y es capaz de levantar cargas relativamente pesadas, teniendo en cuenta su bajo peso. Está compuesto por cuatro ejes y en el extremo de cada uno de ellos se sitúa el conjunto motor-hélice. En multirotores de más de tres ejes, no es necesario utilizar el servo del tricopter. Esto es un valor a tener en cuenta puesto que es una de las piezas que se deterioran con más facilidad.

En un quadcopter no es necesario que los ejes estén situados a 90°, sin embargo es lo más habitual.

Figura 2.2 Ejemplo Quadcopter [22]

En la estructura de la figura superior los ejes forman ángulos distintos a 90°, lo cual se ha diseñado para tener un ángulo de visión mayor en las imágenes obtenidas desde la cámara.

Existen otros tipos de quadcopter, llamados V-Tail los cuales tienen dos motores perpendiculares a la estructura y otros dos con cierta inclinación, lo que permite que los movimientos laterales ser realicen más rápidamente.

A continuación, veremos las dos posibles configuraciones de vuelo de un quadcopter:

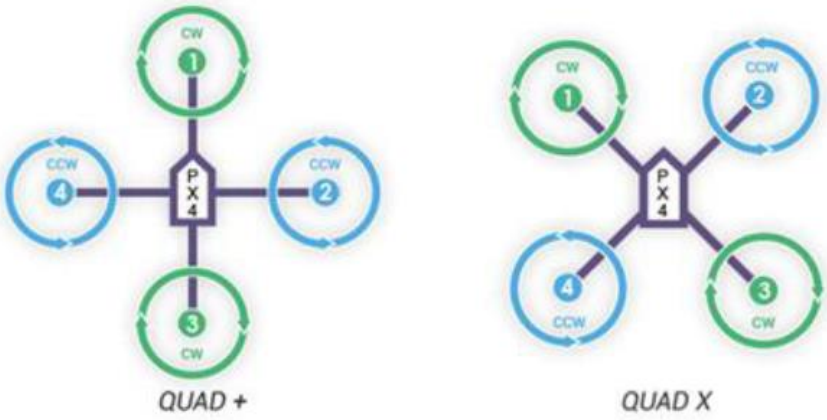

Figura 2.3 Configuración de Quad X vs Quad + [23]

La diferencia entre ambas configuraciones es la posición de los ejes de referencia, es decir dónde está situada la "cabeza". Dependiendo de la configuración de vuelo elegida, el control de vuelo se realizará de forma distinta, ya que para realizar los mismos movimientos, tendrán que rotar los motores de una forma u otra. Éste es un factor importante a tener en cuenta, puesto que el sistema de control depende de la configuración elegida.

Las razones principales para elegir un multirotor de más de 4 ejes son la estabilidad, el peso que es capaz de soportar y la seguridad. Un ejemplo son los hexacópteros, que se utilizan en aplicaciones donde es necesario levantar cargas más pesadas. Además, en caso de fallo de uno de los motores, es posible apagar otro motor de modo que se convierta en un quadcopter y así evitar la caída.

Figura 2.4 Ejemplo Hexacopter [24]

Ésta configuración es muy utilizada en aplicaciones de fotografía y cartografía.

2.1.2 UAV elegido

Finalmente, para la realización de nuestro proyecto hemos elegido un multirotor, ya que el despegue y aterrizaje vertical nos permitirán utilizar nuestro UAV en multitud de aplicaciones. Más concretamente, vamos a utilizar un quadcopter convencional en configuración X, lo cual otorgará al sistema mayor maniobrabilidad y estabilidad, y nos permitirá incluir una cámara en la parte delantera para la captación de imágenes.

Existe una gran variedad de modelos de quadcopter en el mercado, pero hemos decidido realizar el nuestro en una impresora 3D, de modo que podremos ampliarlo y adaptarlo dependiendo de la aplicación que vaya a llevar a cabo. En principio no hemos incorporado ninguna cámara en el quadcopter, aunque en el futuro podemos modificar el diseño e incluir un estabilizador de cámara o gimbal si es necesario. Además, en caso de accidente, es posible imprimir la pieza o piezas que se hayan roto y sustituirlas rápidamente.

2.2 Fundamentos de vuelo

En un cuadricóptero, los movimientos de traslación y rotación se controlan variando la velocidad de rotación de los motores, lo que provoca un cambio en el empuje de los mismos. Es fundamental definir la orientación del UAV, puesto que el modo de vuelo cambiará en función de si utilizamos una configuración en X o en +. Como hemos comentado anteriormente, en el proyecto utilizaremos un Quad-X.

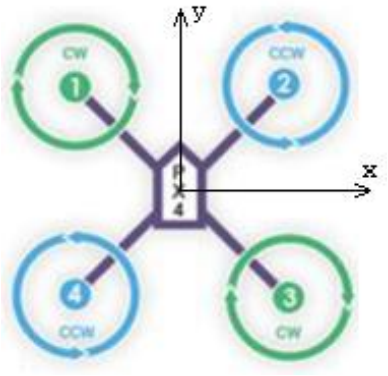

Figura 2.5 Configuración y ejes de Quad-X [23]

Para facilitar la comprensión numeraremos los motores como se indica en la figura superior. La nomenclatura CW o CCW hace referencia al sentido de giro del motor, los motores 1 y 3 giran en sentido de las agujas del reloj y los motores 2 y 4 en el sentido contrario. Esta asignación parece arbitraria pero sirve para compensar el momento cinético del quadcopter.

El movimiento de ascenso o descenso se realiza aumentando o disminuyendo la velocidad de rotación de los cuatro motores a la vez. La traslación y la rotación se realizan generando un empuje diferencial de dos pares de motores. La traslación en el eje x se conoce como movimiento de alebeo o roll y la translación en el eje y se conoce como movimiento de cabeceo o pitch.

Para realizar un movimiento de roll o pitch se ha de producir un aumento de la velocidad de dos motores consecutivos. De este modo para desplazar el quadcopter hacia la derecha habrá que incrementar la velocidad de rotación de los motores 1 y 4 o decrementar la velocidad de los motores 2 y 3. Los movimientos de pitch se realizan de la misma forma, pero con las parejas de motores 1 y 2 frente a 3 y 4.

Por último, para realizar el movimiento de rotación o yaw habrá que incrementar la velocidad de dos motores alternativos, es decir si aumentamos la velocidad de los motores 1 y 3 se producirá una guiñada dextrógira, en cambio si aumentamos la velocidad de los otros dos motores se realizara una guiñada levógira.

2.3 Modelado

Conocer el modelo matemático que representa un sistema es fundamental a la hora de realizar el control, puesto que definirá la estrategia de control. El modelado matemático de un cuadricóptero es complejo puesto que su comportamiento aerodinámico es no lineal y hay que tener en cuenta posibles efectos aerodinámicos del entorno. Sin embargo, este es un tema ampliamente estudiado y se han creado distintos modelos tanto para vuelos interiores [1],[2] en los que no hay efectos del entorno, como para vuelos exteriores [3].

El modelo matemático describe el comportamiento físico de la aeronave a través de una serie de ecuaciones. Para caracterizar un quadcopter a través de su modelo, hay que definir una serie de parámetros relativos a la estructura como son la masa, el centro de gravedad o los momentos de inercia y también una serie de características técnicas relacionadas con los motores como los KV (rpm/V), entre otros. Para realizar el modelado del quadcopter es necesario definir dos sistemas de referencia. El sistema de referencia fijo de la tierra y el sistema de referencia móvil que coincide con el cuerpo del cuadricóptero.

También es posible utilizar el modelado matemático simplificado [4] en el que se incluyen únicamente las dinámicas correspondientes a la orientación del cuadricóptero, se utilizan las ecuaciones de Newton-Euler en las que se asume que el comportamiento del cuadricóptero se corresponde al de un sólido rígido y las ecuaciones de Euler-Lagrange.

Para hacer el modelo más realista habría que incluir efectos aerodinámicos como la fuerza del aire, la resistencia provocada por el rozamiento o el efecto suelo, provocado por el aire que rebota en el suelo y desestabiliza el cuadricóptero al despegar. Sin embargo, debido a las simplificaciones que se realizan, normalmente estos términos se desprecian.

Gracias al programa Matlab, es posible realizar simulaciones tanto con el modelado no lineal completo como con modelo simplificado. De esta forma es posible comprobar si alrededor del punto de trabajo “hover” ambos modelos son igualmente válidos o no.

2.4 Estimación del estado

La estimación del estado del UAV es fundamental en el control del cuadricóptero. Para ello se requiere la utilización de sensores como los giróscopos y los acelerómetros, los cuales están contenidos en la unidad de medición inercial (IMU). Nos encontramos trabajando en una aplicación real, por lo que las medidas no son del todo precisas, es decir presentan ruido. Los valores recibidos de la IMU son de vital importancia puesto que se utilizan en el regulador que estabilizará nuestro UAV.

Aunque la unidad de medición inercial sea de calidad, es necesario filtrar las medidas obtenidas. En este sentido, los filtros más utilizados para este tipo de aplicaciones son el filtro complementario [5] y el filtro Kalman [6], que nos permiten fusionar la información obtenida por los sensores, obteniendo así una estimación de la posición y orientación de la aeronave mucho más precisa. Esto es conocido como data-fusion [7] y adquiere más importancia a medida que aumenta el número de sensores a bordo [8].

2.4.1 Filtro complementario

El filtro complementario es bastante utilizado en aplicaciones de aeronáutica debido a su simplicidad. Este filtro, realiza una ponderación del valor obtenido por dos sensores que se encuentran tomando la misma medida. El factor α puede tomar cualquier valor entre 0 y 1.

$$Valor\ filtrado = Valor_{sensor1} \cdot \alpha + Valor_{sensor2} \cdot (1 - \alpha) \tag{2.1}$$

En nuestro caso lo utilizaremos para filtrar las medidas obtenidas del giróscopo y el acelerómetro+magnetómetro, obteniendo así los ángulos de Euler. La expresión con la que se estimará el ángulo es:

$$Ángulo = \alpha \cdot (Ángulo + gyro \cdot dt) + accel \cdot (1 - \alpha) \tag{2.2}$$

Donde la variable *Ángulo* lleva almacenado el valor del ángulo en el estado anterior, la variable *gyro* tiene unidades de [°/seg], *dt* representa el tiempo transcurrido desde el anterior estado y *accel* es el ángulo obtenido con el acelerómetro. Por último, cabe destacar que para hallar la orientación del UAV se suele utilizar $\alpha \in [0.9 \sim 0.99]$.

De este modo, implementamos un filtro paso banda que eliminará las perturbaciones a altas frecuencias del acelerómetro y el ruido a bajas frecuencias del giróscopo, obteniendo así una medida mucho bastante más realista con un tiempo de estimación muy reducido.

2.4.2 Filtro Kalman

El filtro Kalman [9] es un algoritmo de estimación de estados recursivo, es decir únicamente precisa de las entradas actuales, el estado previo y la matriz de incertidumbre, por lo que es posible utilizarlo en tiempo real. Este filtro trata de estimar el estado minimizando el error cuadrático medio.

El algoritmo está diseñado para trabajar con un sistema dinámico lineal, pero es posible aplicarlo a sistemas no lineales utilizando una modificación del mismo llamada filtro extendido de Kalman. Actualmente se utiliza en múltiples aplicaciones como sistemas de navegación, control de vehículos, procesamiento de datos, aeronáutica, etc.

El algoritmo de Kalman consta dos procesos, el proceso de *Predicción* donde se predice el estado y la covarianza del error y el proceso de *Corrección* en el cual se calcula la ganancia y se corrige la covarianza y la estimación anterior.

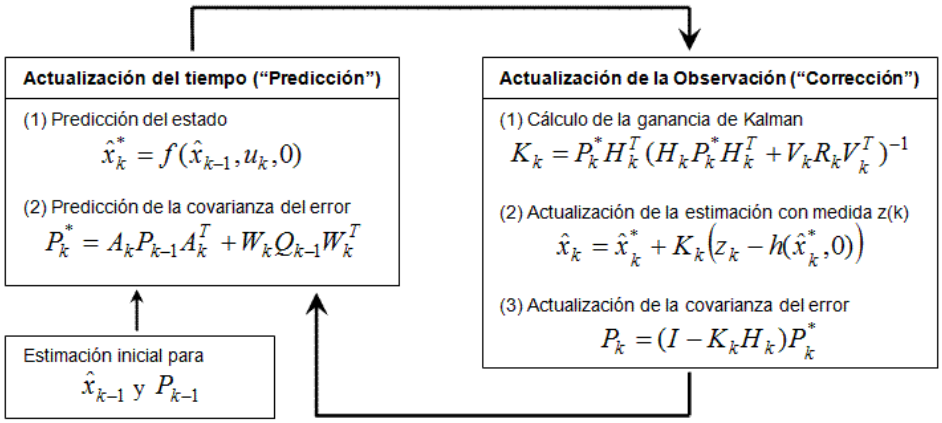

Figura 2.6 Procesos del Filtro de Kalman [25]

Para entender el funcionamiento del filtro de Kalman debemos comprender la nomenclatura utilizada en los desarrollos matemáticos:

ϕ_k : Matriz de transición de estados

$X_{k|k-1}$: Estimado a priori del vector de estados

$P_{k|k-1}$: Covarianza del error asociada a la estimación a priori

z_k : Vector de mediciones en el tiempo k

H_k : Matriz que relaciona las mediciones con el vector de estados

R_k : Matriz de covarianza del ruido de las mediciones

Los cálculos matemáticos que se realizan en el proceso de Predicción son:

Estimación a priori:

$$\hat{X}_{k|k-1} = \phi_k \cdot X_{k-1|k-1} \tag{2.3}$$

Covarianza del error asociado a dicha estimación:

$$P_{k|k-1} = \phi_k \cdot P_{k-1|k-1} \cdot {\phi_k}^T + Q_k \tag{2.4}$$

En el proceso de Corrección los cálculos son:

Actualización de las mediciones:

$$y_k = z_k - H_k \cdot \hat{X}_{k|k-1} \tag{2.5}$$

Calculo de la ganancia de Kalman

$$K_k = P_{k|k-1} \cdot {H_k}^T \cdot (H_k \cdot P_{k|k-1} \cdot {H_k}^T + R_k)^{-1} \tag{2.6}$$

Estimación a posteriori:

$$\hat{X}_{k|k} = \hat{X}_{k|k-1} + K_k \cdot y_k \tag{2.7}$$

Covarianza del error asociado a la estimación a posteriori:

$$P_{k|k} = P_{k|k-1} \cdot (I - K_k \cdot H_k) \tag{2.8}$$

Las estimaciones a posteriori son los valores que obtenemos después de aplicar el filtro Kalman a las medidas recogidas por los sensores.

La principal ventaja de este tipo de filtros es que realizan una estimación del estado muy precisa y únicamente necesita almacenar el estado anterior, por lo que el coste computacional no es demasiado grande.

Con el fin de comparar la efectividad del filtro complementario con respecto al filtro extendido de Kalman, se muestra la "Figura 2.7" recogida del trabajo [10]. En ella se muestra la estimación del ángulo de alebeo o roll utilizando ambos métodos de estimación.

En rojo se presenta la referencia, es decir el ángulo real del UAV, en verde el ángulo estimado utilizando el filtro de Kalman y en azul el ángulo estimado mediante el filtro complementario.

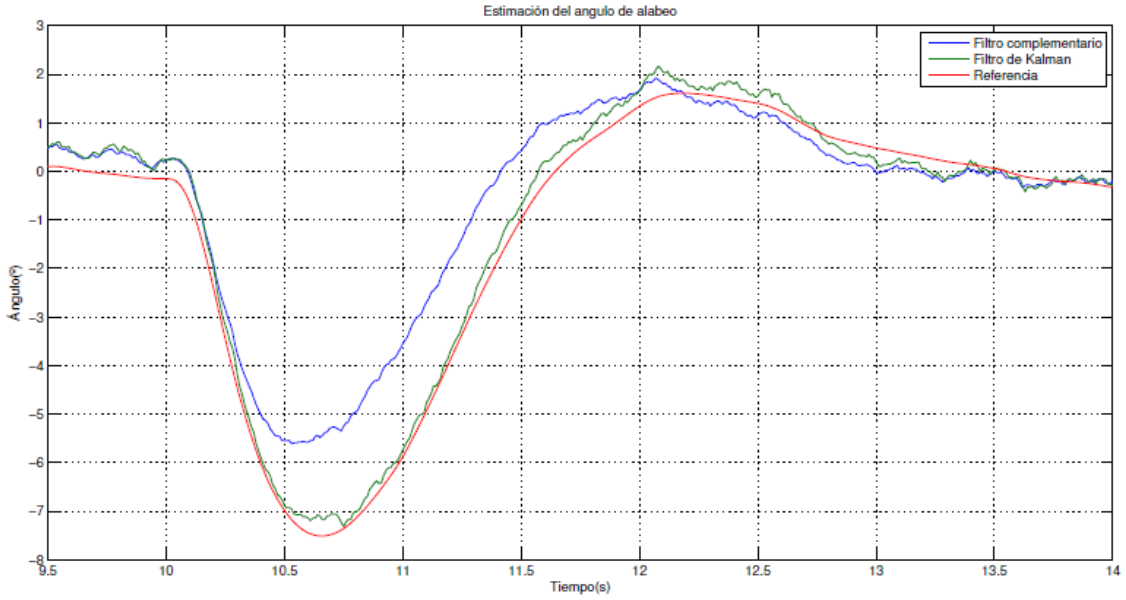

Figura 2.7 Filtro complementario vs filtro Kalman [10]

Como podemos observar, ambos métodos de estimación del ángulo siguen la referencia correctamente, sin embargo, el filtro de Kalman resulta más efectivo cuando se producen rápidos cambios en la referencia. El resto del tiempo, ambos métodos aportan ángulos prácticamente iguales. Por otro lado, el filtro extendido de Kalman es un método estadístico, más complejo que requiere un tiempo de procesamiento superior y cierta capacidad de procesamiento, en cambio el filtro complementario tiene una lógica muy sencilla lo que disminuye dicho tiempo. En definitiva, ambos son buenos métodos de estimación de estado, razón por la cual en la literatura podremos encontrar multitud de sistemas en los que se han utilizado.

2.5 Estrategias de control

Durante los últimos años, con la irrupción de los cuadricópteros y multirotores en general, se han desarrollado multitud de reguladores y algoritmos de control. La gran mayoría de ellos se apoyan en modelos matemáticos complejos por lo que para diseñar el sistema de control es crucial tener en claro el objetivo de la regulador y no perderse en los desarrollos matemáticos.

El principal objetivo del control es estabilizar el cuadricóptero. Como hemos visto anteriormente, un multirotor tiene un comportamiento dinámico no lineal y multivariable por lo que es fundamental que nuestro sistema de control sea lo suficientemente rápido para controlar nuestro sistema sin llegar a la sobreoscilación.

Actualmente, existen distintas estrategias de control como el regulador de Holger Voos [1] basado en el control por linealización en la realimentación, el cual defiende que en los UAV existen dos lazos principales de control. El lazo interno de control del vehículo, responsable de la estabilización y el lazo externo al que denomina "mission control loop" que regula la velocidad y calcula la trayectoria en función de unos comandos o unos puntos marcados. El control de la actitud del lazo interno se realiza mediante linealización en la realimentación que tiene en cuenta la dinámica no lineal del cuadricóptero. Para el control de la velocidad se utiliza un regulador proporcional.

En la imagen siguiente se representa la estructura anidada de control en la que se aprecian los dos lazos. Donde ẋ, ẏ, ż son las velocidades lineales realimentadas en el lazo externo y donde ϕ, θ, ψ son los ángulos asociados al roll, pitch y yaw.

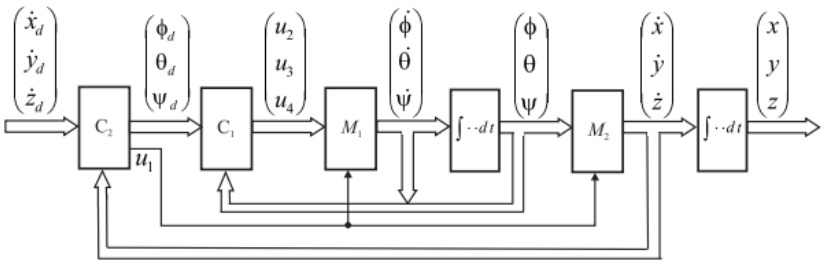

Figura 2.8 Lazos del sistema de control por linealización en la realimentación [1]

El lazo interno es mucho más rápido que el externo puesto que debe estabilizar el cuadricóptero en los ángulos deseados por el bucle externo. Con el fin de obtener las medidas de posición, orientación y velocidad necesarias, hace uso de una IMU e implementa un sistema estimación de estados basado en un filtro Kalman. Para ver las ecuaciones que representan este regulador consultar la referencia.

Otro sistema de control utilizado en este tipo de aplicaciones es el control LQR (*Linear Quadratic Regulator*) como el implementado en el artículo de Bouabdallah [11] en el que se compara un regulador LQ con un PID. Un regulador LQ está basado en la teoría de control óptimo por la cual se describen las ecuaciones de las variables de control que minimicen el coste funcional.

Es decir, se utiliza un regulador LQ para minimizar las desviaciones de altitud y orientación del cuadricóptero, ajustando unos valores de ponderación hasta obtener el regulador óptimo. Por tanto este tipo de regulador precisa realizar varias simulaciones para ajustarlo.

Para diseñar el sistema de control LQR el primer paso es pasar las ecuaciones linealizadas entorno al punto de trabajo hover, a la forma del espacio de estados [12]. De este modo, el vector de estado x está formado por la posición p, la actitud q, la velocidad lineal v y la velocidad angular w. Cabe destacar que en el sistema propuesto [12] la actitud está expresada en cuaternios. Los parámetros de control elegidos son la posición p y la actitud q. El controlador tendrá la siguiente expresión.

$$u(t) = -K \cdot \Big(x(t) - x_{ref}(t)\Big) \tag{2.9}$$

Minimizando la expresión de la función del coste obtendremos la ganancia K.

$$J = \int_0^\infty \Big[\Big(x(t) - x_{ref}(t)\Big)^T \cdot Q \cdot \Big(x(t) - x_{ref}(t)\Big) + u(t)^T \cdot R \cdot u(t)\Big]\, dt \tag{2.10}$$

En la imagen siguiente se explica el funcionamiento del control LQR utilizando cuaternios.

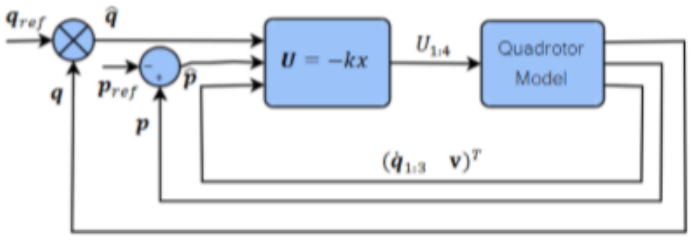

Figura 2.9 Control LQR con cuaternios [12]

El problema de estos tipos de reguladores que un cambio en los parámetros de control no se ve claramente reflejado en un cambio en el funcionamiento del sistema. Sin embargo, utilizando reguladores PID es sencillo ver el efecto que produce cambiar una constante de control.

Un regulador PID es un sistema de control por realimentación que se utilizan en multitud de aplicaciones industriales debido a su sencillez y a su flexibilidad. Estos reguladores constan de tres parámetros, Proporcional, Integral y Derivativo, cuyos efectos en el control del sistema explicaremos más adelante. Dependiendo de la aplicación será más adecuado utilizar un regulador de tipo P, PI, PD o PID. Básicamente, un regulador PID calcula y ajusta el error cometido entre el valor medido y el valor deseado.

La selección de las constantes de control óptimas no es un proceso sencillo. Se han realizado varios estudios para encontrar el procedimiento más adecuado para obtener respuestas rápidas con la mínima sobreoscilación posible [13]. Por un lado, el ajuste manual del regulador implica la necesidad de realizar un gran número de simulaciones y ensayos para juzgar su comportamiento, lo que nos puede llevar un tiempo y esfuerzo considerables. Por otro lado, aunque existen técnicas de optimización de los parámetros del PID convencional, éstas requieren un gran coste computacional y son difíciles de generalizar. Por tanto, habrá que diseñar algoritmos de regulación PID robustos como el creado por B. Kada en [14] .

También es posible utilizar sistemas de control de lógica difusa, como el expuesto en [15], para crear un regulador PID que ajusta automáticamente las constantes de control. El sistema propuesto se denomina "*Self-Tuning Fuzzy PID Controller*" y se basa en el un sistema de inferencia Fuzzy para encontrar los valores más adecuados. En la siguiente figura se muestra el funcionamiento general del sistema.

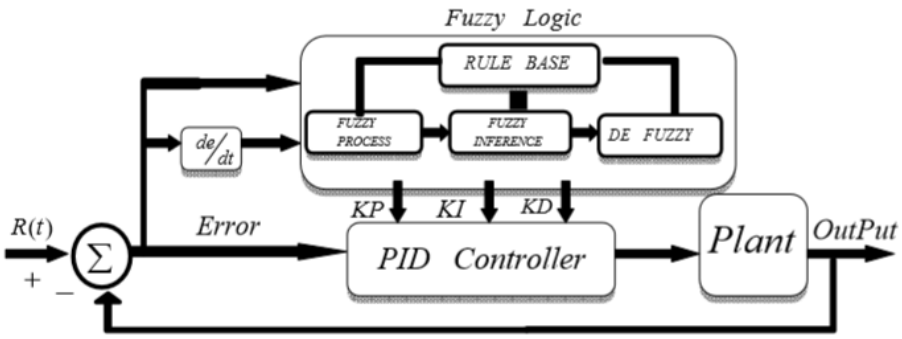

Figura 2.10 Diagrama de funcionamiento del "Self-Tuning Fuzzy PID Controller" [15]

Como podemos observar, las entradas del sistema fuzzy son el error e(t) y el gradiente de dicho error $\frac{de(t)}{dt}$. Éstas señales se convierten a parámetros fuzzy, tras lo cual se realiza el proceso de inferencia con el que obtendremos las salidas K_p, K_i y K_d. Los procesos del sistema fuzzy se basan en unas reglas para asignar los valores a las salidas en función de las entradas. Estas reglas se formulan gracias a la experiencia y conocimiento del diseñador y pueden sufrir modificaciones dependiendo del UAV a controlar.

Además, en [15] se han implementado en Simulink un regulador PID convencional, un regulador LQR, un controlador de lógica Fuzzy y el "Self-Tuning Fuzzy PID Controller" para analizar qué sistema de control ofrece una respuesta más adecuada. Con todos ellos se consigue una regulación muy buena que cumple con los objetivos marcados, sin embargo el "Self-Tuning Fuzzy PID Controller" otorga una respuesta mejor, con un tiempo de establecimiento muy bajo y sin overshoot.

Aunque la lógica difusa fue formulada en 1965, actualmente se están desarrollando distintos sistemas de control basados en ella. Éste desarrollo se ha motivado en parte gracias a la disponibilidad de la herramienta Fuzzy Logic implementada Matlab. En este sentido, se han desarrollado sistemas de control fuzzy inteligente [16].

Otra forma de afrontar el control del UAV es mediante la técnica de control H_∞ utilizada en [17] o como en [18] donde se propone crear un controlador robusto capaz de mantener el vuelo en entornos difíciles en los que el viento pueda crear turbulencias y en el que las medidas de los sensores contengan ruido.

Por último, una aproximación más novedosa es el uso de redes neuronales que aprendan las dinámicas del quadcopter mediante entrenamiento offline con datos de vuelo obtenidos previamente [19]. También es posible ir un paso más allá utilizando redes neuronales en un observador para estimar el estado del UAV y para generar señales de control [20]. De esta forma, es posible aprender las dinámicas del quadcopter durante el vuelo, incluyendo efectos como la fricción aerodinámica o el blade-flapping.

Capítulo 3 Selección de los elementos

3.1 UDOO

Figura 3.1 UDOO Quad [26]

UDOO Quad es un dispositivo de hardware libre que funciona como un ordenador y que aúna en una sola placa la potencia del procesador Freescale i.MX6 que trabaja con Linux, Android o Google ADK y la flexibilidad del Arduino que funciona gracias al procesador Atmel SAM3X8E.

Se trata de una placa encaminada a la realización de prototipos para el diseño y desarrollo de software. UDOO es una herramienta que permite el desarrollo de distintos proyectos sin necesidad de ser un experto en la materia. Además, tiene un gran potencial educativo para la realización de proyectos Do-It-Yourself.

El procesador i.MX6 soporta la reproducción de videos en 1080p con bajo consumo y contiene un motor de gráficos 3D capaz de capaz de proporcionar hasta 200Mt/s. En esta línea, podemos afirmar que la UDOO Quad tiene el procesamiento de cuatro Raspberri Pi básicas y la versatilidad del Arduino Due.

Es posible utilizar ambos procesadores para que trabajen conjuntamente realizando las tareas más pesadas en la parte de Linux, como por ejemplo, el procesamiento de las imágenes captadas por la cámara del quadcopter, a la vez que se realiza el control de éste en la parte del Arduino. También es posible utilizar esta funcionalidad ejecutando una aplicación en el móvil para que se comunique con el procesador i.MX6 que estará funcionando con Android y controlar así la parte de Arduino.

Por otro lado, UDOO permite cambiar en pocos segundos entre Linux o Android remplazando la tarjeta Micro SD y reiniciando el sistema. De este modo, es posible realizar varios proyectos en distintos lenguajes en una sola placa.

En definitiva, lo que nos aporta la placa UDOO Quad es una gran potencia de procesamiento y una gran flexibilidad, que nos permitirá utilizarla en multitud de aplicaciones, desde el reconocimiento facial hasta el control de un quadcopter, pasando por aplicaciones de domótica o procesamiento de datos.

Las características principales y especificaciones son las detalladas a continuación.

- Freescale i.MX 6 ARM Cortex-A9 CPU Quad core 1GHz
- Atmel SAM3X8E ARM Cortex-M3 CPU (el mismo que tiene el Arduino Due)
- Gráficos integrados, cada procesador proporciona tres aceleradores por separado para 2D, OpenGL® ES2.0 3D and OpenVG™
- 1GB RAM DDR3
- Micro SD
- HDMI y conexión para pantalla táctil LVDS
- 76 GPIO totalmente accesibles
- Módulo WiFi
- Ethernet RJ45 (10/100/1000 MBit)
- Conexiones para cámara UDOO y pantalla táctil LVDS
- USB tipo A (x2), Mini USB y Mini USB OTG
- Entradas de Audio y Micrófono
- Conector SATA
- Alimentación a 12V y conector externo de batería

Como hemos comentado anteriormente, el presente proyecto trata de diseñar e implementar el sistema de vuelo de un quadcopter, pero éste no es un proyecto aislado, sino que se utilizará en otros proyectos de mayores dimensiones por lo que debe ser capaz de realizar, entre otras cosas, procesamiento de imagen. Para ello, es necesario disponer de una unidad de procesamiento muy potente.

Debido a la morfología del quadcopter y a que el modelo del comportamiento aerodinámico es no lineal, es necesario incluir distintos elementos electrónicos para controlarlo. Elementos como ESC *(Electronic Speed Controllers)*, sensores inerciales, receptores RC, módulos GPS o cámaras. Por tanto, es necesario disponer de una placa controladora con gran cantidad de GPIO's, así como puertos USB y periféricos encaminados a la comunicación (USART, SPI, I2C) para interconectar los distintos elementos que componen dicho quadcopter.

Por otro lado, las dimensiones físicas de la placa controladora son de 11 x 8.5 cm por lo que no habrá problema en integrar este sistema a bordo. Las dimensiones físicas son importantes, sin embargo el factor limitante es el peso puesto que a mayor peso, peor será el comportamiento del quadcopter durante el vuelo. Puesto que el peso de la UDOO es de 135g, podemos permitirnos utilizarla sin que suponga un perjuicio.

Actualmente existen gran cantidad de placas controladoras comerciales como Naza, OpenPilot, KK, etc. Sin embargo la gran mayoría son totalmente cerradas, es decir, son cajas negras inaccesibles y por tanto, inadecuadas para proyectos de este tipo.

La UDOO tiene un peso ligeramente superior a las placas controladoras mencionadas anteriormente, pero también posee un procesamiento y flexibilidad superiores. Además, es una controladora totalmente abierta que trabaja con Arduino, un lenguaje de programación open-source para el que se dispone de multitud de ejemplos.

En definitiva, hemos seleccionado la UDOO debido a la potencia y versatilidad que la caracterizan y a las dimensiones y bajo peso que la hacen adecuada para el control sobre multicópteros.

3.2 Motores y hélices

Los motores más utilizados en aplicaciones sobre UAV son los motores eléctricos brushless, es decir, sin escobillas. En este tipo de motores la corriente pasa directamente por las tres fases de los bobinados del estator, activándose una fase después de otra, lo cual genera un campo electromagnético que interacciona con el campo magnético de los imanes del rotor. De este modo, aparece una fuerza que hace girar el motor.

En el interior del motor brushless, hay unos sensores de efecto Hall que detectan la posición del motor en cada momento e informan al driver o ESC de ello. Al no tener escobillas ni colector, será necesario utilizar dicho ESC para controlar la velocidad de giro del motor.

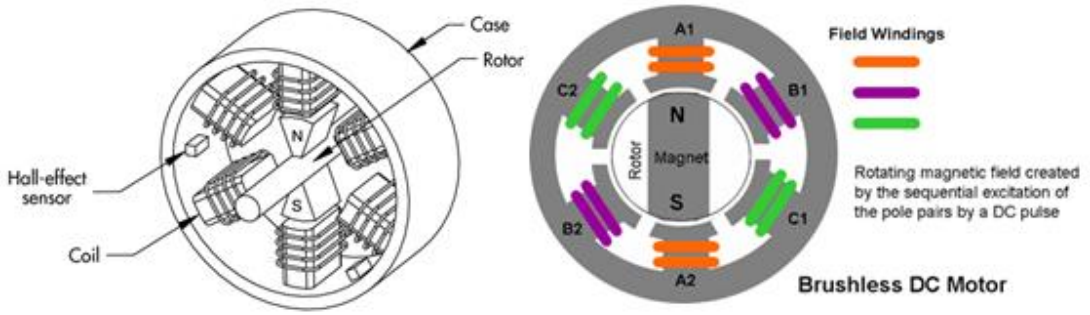

Figura 3.2 Morfología motor Brushless DC [27]

Utilizamos este tipo de motores puesto que tienen las siguientes ventajas:

- Conmutación electrónica basada en los sensores de efecto Hall
- Gran rango de velocidades (sin estar limitada por los cepillos)
- Alta eficiencia (no hay caída de voltaje en los cepillos)
- Tamaño y peso reducidos
- Baja generación de ruido eléctrico

Sin embargo, también tiene sus desventajas:

- Mayor coste y complejidad
- Necesita un ESC para mantener el motor en marcha

Estas son las razones fundamentales por las que hemos elegido un motor brushless en lugar de uno brushed. Por otro lado, a la hora de elegir el modelo adecuado para nuestro quadcopter tenemos que tener otra cosa en cuenta, los kV. Éste parámetro nos indica el número de revoluciones por minuto (rpm) a la que es capaz de girar por cada voltio entregado.

No obstante, el par entregado es inversamente proporcional a los kV, por tanto, a mayor kV aumentarán las rpm y disminuirá el par entregado. Lo cual nos lleva a buscar un compromiso entre velocidad a la que queremos que gire y el par que necesitamos que entregue.

Otros factores que debemos tener en cuenta a la hora de elegir el modelo son la relación potencia-peso y la relación empuje-peso. A continuación expondremos las características del motor elegido.

Tabla 3.1 Características motor Turnigy L2210C

Turnigy L2210C Brushless Motor	
Dimensiones	28mmx25mm
Peso	63g
Kv	1200
Voltaje de trabajo	7.2V - 11.1V
Potencia máxima	150W
Corriente máxima	15.8A
Empuje	700g
Diámetro del eje	3mm

Por tanto las relaciones potencia-peso y empuje-peso serán:

$$\frac{\mathrm{P}}{\mathrm{m}} = \frac{150}{63} = 2.38\ \mathrm{W/g} \qquad \frac{E}{m} = \frac{700}{63} = 11.1\ g/g \tag{3.1}$$

Los cuales son bastante buenas, comparándolas con otros motores del mercado. Estas relaciones dan información, pero debemos asegurarnos de que los motores elegidos son capaces de levantar con suficiente facilidad toda la estructura, incluyendo todos los elementos. Para que el quadcopter vuele con suficiente fluidez, tendremos que asegurar que el empuje total sea:

$$\boxed{E_T > 1.5 \cdot \left(P_{struct} + 4 \cdot P_m + 4 \cdot P_{ESC} + P_{UDOO} + P_{bat} + P_{IMU} + P_{receptor\ RC}\right)} \quad (3.2)$$

Nomenclatura:

$E_T \equiv$ Empuje total de los motores

$P_{struct} \equiv$ Peso de la estructura

$P_m \equiv$ Peso del motor+hélice

$P_{ESC} \equiv$ Peso del variador electrónico de velocidad

$P_{UDOO} \equiv$ Peso de la placa UDOO

$P_{bat} \equiv$ Peso de la batería

$P_{sensores} \equiv$ Peso de todos los sensores

En la siguiente tabla se muestran los pesos de los elementos seleccionados:

Tabla 3.2 Pesos de los elementos del UAV

P_{struct}	P_m	P_{ESC}	P_{UDOO}	P_{bat}	P_{IMU}	$P_{receptor\ RC}$
412g	75g	15g	135g	420g	3g	18g

Evaluamos la expresión (3.2) teniendo en cuenta los datos de la tabla anterior:

$$\frac{E_T}{P_T} = \frac{2800}{1348} = 2{,}08 \quad (3.3)$$

El empuje es 2,08 veces mayor que el peso del UAV por lo que hemos seleccionado un motor adecuado para nuestro sistema.

Por último, debemos elegir el tipo de hélice que montaremos en los motores. Existen varios tipos de hélices pero podemos distinguir dos grandes tipos según su morfología:

- **Hélices de dos palas:** Son las hélices utilizadas habitualmente en aplicaciones de radio control. Son más eficientes que las de tres palas a las velocidades y tamaños de hélices que utilizamos regularmente.
- **Hélices de tres palas:** Se usan habitualmente cuando se tiene un límite de diámetro de la hélice.

Puesto que el chasis del UAV elegido no supone ninguna limitación del diámetro de la hélice, es decir hay suficiente espacio, hemos elegido las hélices de dos palas.

Ahora debemos elegir el material de dicha hélice, los dos materiales más utilizados son el plástico y la fibra de carbono. El empuje es el mismo para hélices de la misma forma, sin embargo el peso de las de fibra de carbono es notablemente inferior. Por otro lado, el precio de éstas últimas es mayor.

Como hemos visto anteriormente, el peso total del quadcopter es un parámetro crítico, sin embargo el empuje total es mucho mayor que el peso total, por tanto para nuestro proyecto utilizaremos hélices de plástico que nos permitirán abaratar los costes sin que afecte al buen funcionamiento del UAV. A continuación se muestra el tipo de hélice que vamos a utilizar:

Figura 3.3 Hélice de dos palas [28]

Las hélices seleccionadas son de 9x4.7, es decir, sus dimensiones son de 9 pulgadas con un paso de la hélice de 4.7. Hay que tener en cuenta que al adquirir las hélices que para compensar el momento cinético del quadcopter al volar es necesario hacer girar dos hélices en un sentido y otras dos en el sentido contrario. Por tanto tenemos que pedir dos hélices CW y otras dos CCW.

3.3 ESC

Un ESC (*Electronic Speed Controller*, por sus siglas en inglés), es un dispositivo electrónico capaz de variar la velocidad de un motor eléctrico. Por eso se les conoce comúnmente como variadores. Se utilizan habitualmente en aplicaciones de radio control para controlar motores eléctricos brushless.

Básicamente, un ESC entrega una potencia de salida de corriente alterna y trifásica, la cual está limitada por la tensión de entrada al variador, para controlar el motor enviando una secuencia de señales de corriente alterna. El variador necesita una tensión de alimentación a la entrada, por lo que los ESC modernos incluyen un módulo BEC (*Battery Eliminator Circuit*) que regula la tensión de entrada al variador, ahorrándonos utilizar otra batería.

Sin embargo, no todos los variadores traen el BEC integrado por lo que habrá que tenerlo en cuenta a la hora de seleccionar el ESC. Cabe destacar que dependiendo de las especificaciones del modelo seleccionado, podremos utilizar una batería u otra. Además debemos tener en cuenta la corriente máxima que es capaz de entregar al motor.

El ESC elegido es el siguiente, cuyas características técnicas son:

Tabla 3.3 Características ESC Turnigy

Turnigy Multistar 20A Slim BLHeli ESC	
Corriente constante	20A
BEC	5V / 3A
Celdas batería LiPo	2-4
Celdas batería NiMH	5-12
Peso	20.3g
Dimensiones	62x13x8 mm
Frecuencia de trabajo max	480 Hz

A la vista de estas características, podemos afirmar que el variador será capaz de entregar la corriente demandada por el motor, puesto que tanto la corriente media como la corriente máxima del ESC son superiores a las del motor. Por tanto, podemos afirmar que el variador seleccionado es compatible con el motor brushless Turnigy L2210C ya que es capaz de entregar holgadamente la potencia demandada por el motor.

Habría sido posible elegir un ESC de 20 A o 30 A, cuya corriente media es bastante superior a la demandada por el motor, sin embargo el peso sería mayor y no estaríamos aprovechando todo el potencial del variador.

3.4 Batería

Existen distintos tipos de baterías, pero las más utilizadas en aplicaciones de UAV son las de polímero de litio, conocidas como LiPo. Existen otros tipos de baterías como las de NiMH o las de NiCd, sin embargo hay tres características fundamentales que hacen a las LiPo más adecuadas para este tipo de aplicaciones:

- Son ligeras y se pueden fabricar de distintas dimensiones.
- Tienen una gran capacidad en un tamaño reducido.
- La tasa de descarga es alta, ideal para alimentar sistemas eléctricos muy exigentes en corriente.

Este tipo de baterías se utilizan en multitud de dispositivos como teléfonos móviles u ordenadores, pero en UAV donde se demandan grandes picos de corriente, necesitamos tasas de descarga superiores.

Las principales desventajas de estas baterías son que es necesario un mantenimiento adecuado, especialmente en los procesos de carga y descarga puesto que está directamente relacionado con la vida útil de la batería, y que son peligrosas debido a que son altamente inflamables y si no se realiza un uso adecuado de ellas, pueden llegar a incendiarse o explotar.

A la hora de elegir la batería LiPo debemos tener en cuenta tanto lo expuesto anteriormente, como el tipo de batería que el variador es capaz de soportar. Es decir, este tipo de baterías se distinguen entre ellas principalmente por el número de celdas que contienen.

El número de celdas va directamente relacionado con la tensión que es capaz de entregar. De este modo, existen baterías de dos celdas (2S: 7.4V), de tres celdas (3S: 11.1V), de cuatro celdas (4S: 14.8V) y así sucesivamente. También hay que tener en cuenta que si el número de celdas o capacidad de ellas aumenta, el peso también aumentará.

Para nuestro quadcopter es necesario incluir dos baterías, una batería de alta capacidad y alta tasa de descarga para alimentar los cuatro motores, y otra más pequeña para alimentar la UDOO, puesto que precisa de una alimentación externa para funcionar. Los modelos elegidos son los siguientes:

Figura 3.4 Batería LiPo RCInnovations [29]

Cuyas características se pueden resumir en la siguiente tabla:

Tabla 3.4 Características batería RCInnovations

RCInnovations Desire power V8 Series Battery	
Numero de celdas	3
Voltaje	11.1 V
Capacidad	6000 mAh
Tasa de descarga media	30 C (180 A)
Tasa de descarga máxima	60 C (360 A)
Peso	420 g
Dimensiones	36x45x136 mm

Hemos elegido esta batería de tres celdas porque, como vemos en sus características, tiene una alta capacidad (6000mAh) lo que nos dará más tiempo de vuelo y porque es capaz de entregar grandes corrientes, puesto que su tasa de descarga media es de 30 C (180A). Además, su peso es de 420 g por lo que la relación capacidad peso será de:

$$\frac{C}{m} = \frac{6000}{420} = 14.29\ mAh/g \tag{3.4}$$

Una relación bastante buena, comparándola con otras baterías del mercado.

La batería elegida para alimentar la UDOO es de tres celdas, puesto que, si miramos en las características, la tensión de alimentación de dicha placa es de [6V- 15 V]. Puesto que la UDOO no demandará grandes picos de corriente, el factor más relevante es el peso, por lo que hemos elegido la batería parrot ARdrone 2.0 de 1500 mah de capacidad, una corriente máxima de 1.5 A y un peso de 120g.

Figura 3.5 Batería ARDrone 2.0 [39]

3.5 Transmisor y receptor de radio control

Para realizar el control remoto del UAV, es necesario utilizar un transmisor de radio control a través del cual mandamos los comandos de los ángulos deseados. El transmisor de radio control es un elemento independiente, es decir, no debemos consultar las características y elegir un modelo u otro en función de los elementos seleccionados anteriormente. Existen transmisores de radio control que trabajan a distintas frecuencias pero los utilizados habitualmente trabajan a una frecuencia de 2,4 GHz.

Cualquier pareja transmisor-receptor funcionará correctamente por sí solo, por lo que únicamente tendremos que tener en cuenta las funcionalidades que nos aporta.

Figura 3.6 Transmisor Turnigy 9X [30]

En este sentido, hemos elegido es el modelo Turnigy 9X porque tiene nueve canales, cuatro de ellos están encaminados a controlar el vuelo de quadcopter y el resto de ellos quedan libres para controlar otros dispositivos como cámaras o para cambiar el modo de vuelo a un control más agresivo o a uno más estable.

Además este modelo de transmisor tiene una pantalla a través de la cual es posible cambiar ciertos valores de la configuración o guardar distintas configuraciones del mando. Esto nos aportará gran flexibilidad a la hora de realizar las pruebas de vuelo.

3.6 Unidad de medición inercial (IMU)

La IMU o unidad de medición inercial, es un dispositivo electrónico compuesto de una serie de sensores como giróscopos o acelerómetros a través de los que podemos determinar la posición y orientación instantánea del cuadricóptero.

Es uno de los elementos más importantes que componen el UAV ya que cuanto más precisa sea la medida de la orientación, menos error estaremos cometiendo y por tanto más fácil será controlarlo. Desde el punto de vista del control, de nada sirve tener un regulador excelente si la medida que recibimos de la IMU no se corresponde con la realidad. Por eso, estos dispositivos tienen más grados de libertad que variables, es decir son dispositivos redundantes.

Queremos medir la orientación en los tres ejes que definen el sistema:

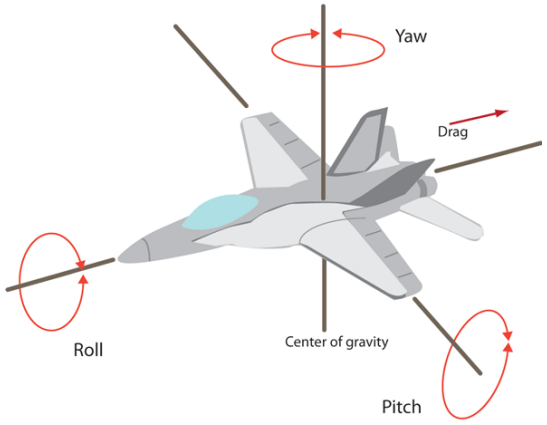

Figura 3.7 Ejes de referencia en un UAV [31]

En principio, eso sería posible obtenerlo utilizando únicamente 3 giróscopos, uno para cada eje. La medida que se obtiene de los giróscopos es precisa, sin embargo son muy susceptibles a las vibraciones y si solo utilizamos los giróscopos se crea una deriva, es decir se va acumulando el error y la medida se desfasa unos cuantos grados del valor real. Este fenómeno se conoce como “drift”.

En un quadcopter, las vibraciones producidas por los motores no son despreciables, por tanto será necesario incluir otros sensores como acelerómetros para compensar dicho efecto. Los acelerómetros son menos precisos que los giróscopos pero no acumulan error a largo plazo, por lo que la pareja giróscopos-acelerómetros se utiliza en todas las IMU de calidad. Sin embargo, como la guiñada o yaw es ortogonal a la gravedad, el acelerómetro no funcionará perfectamente y necesitaremos otro elemento para ajustar el efecto del giróscopo.

Para ello algunas IMU incorporan magnetómetros, que combinados con los acelerómetros y giróscopos, nos permiten calcular una medida muy fiable y precisa de la orientación.

En definitiva, la unidad de medición inercial tendrá un grado de libertad (GL) por cada giróscopo, acelerómetro o magnetómetro. En otro tipo de aplicaciones, será suficiente utilizar una IMU de 3GL (giróscopos), pero para su uso en UAV se utilizan al menos de 6GL (giróscopos + acelerómetros).

Para nuestro proyecto hemos seleccionado el modelo *Adafruit 10-DOF IMU Breakout* que contiene tres giróscopos, tres acelerómetros, tres magnetómetros y un sensor de presión BMP180, con el cual podemos determinar la altura.

Las características técnicas de los sensores son las siguientes:

Tabla 3.5 Características Adafruit 10 DOF IMU

Adafruit 10-DOF IMU Breakout	
Tensión de alimentación	3V / 5V
Escalas del giróscopo	±250, ±500, ±2000 °/s
Escala del campo magnético	±1.3 ~ ±8.1 Gauss
Escala del acelerómetro	±2g, ±4g, ±8g, ±16g
Dimensiones	38x23x3 mm
Peso	2.8 gramos

Como podemos observar en la imagen, la IMU utiliza la comunicación I2C por lo que podremos conectarlo únicamente con 3 cables, y la tensión de alimentación es de 3V por lo que la UDOO podrá leer perfectamente la información recibida en los pines. Recordemos que si se escribe información en los pines de Arduino utilizando 5V de alimentación, se podría romper la placa.

Por último, cabe destacar que para traducir la información recibida en orientación es necesario realizar una serie de cálculos y habrá que estimar el ángulo mediante el uso de un filtro complementario o, si es necesario, implementar un filtro Kalman del que hemos hablado en el estado del arte.

Capítulo 4 Integración de los elementos del Quadcopter

4.1 Receptor RC

4.1.1 Conexionado del receptor RC y canales

Como hemos comentado anteriormente, vamos a utilizar el receptor de RC Turnigy 9x, que contiene nueve canales. Cada canal está asociado a un stick o a un interruptor del transmisor de radio control, por lo que es imperativo conocer qué información recibe cada uno.

Cuatro de ellos están asociado a los movimientos de vuelo (roll, pitch, yaw y throttle), otros cuatro quedan libres para controlar diferentes dispositivos (estabilizadores para cámaras) o para alterar el estilo de vuelo del cuadricóptero, y por último el canal de la batería, por donde se alimentará el dispositivo.

En principio, todos los receptores de RC deben tener la misma asignación de canales, sin embargo en la práctica no siempre es así, por lo que es vital comprobarlo. Una forma muy sencilla es utilizar un servo y conectarlo directamente al receptor RC, comprobando así cada uno de los canales.

Una vez realizado este proceso, debemos asignar un pin de entrada a cada uno de los canales que vayamos a utilizar. De momento sólo nos interesa obtener los datos del roll, pitch, yaw y throttle puesto que estamos trabajando en el control del cuadricóptero en sí. Hemos asignado los pines de la siguiente manera:

- *Roll_In_Pin* → *2*
- *Pitch_In_Pin* → *3*
- *Throttle_In_Pin* → *4*
- *Yaw_In_Pin* → *5*

Que corresponde con la asignación de los pines de entrada del Arduino. Si quisiéramos conectar el resto de los canales, realizaríamos el mismo procedimiento. Una vez que hemos conectado y alimentado el receptor de radio control, podemos pasar a la lectura de los datos.

4.1.2 Lectura de datos

Existen dos formas de transmitir la información contenida en el receptor de radio control, utilizando una señal PPM (Pulse Position Modulation) o utilizando una señal PWM (Pulse Width Modulation).

El receptor de radio control seleccionado utiliza la modulación PWM para transmitir la información. En las señales PWM, se modifica el ciclo de trabajo $D=\frac{T_{on}}{T}$ de una señal periódica para transmitir la información. Es decir, existe una relación entre el ancho del pulso y el valor transmitido. El concepto de PWM se puede explicar fácilmente, mirando la siguiente ilustración.

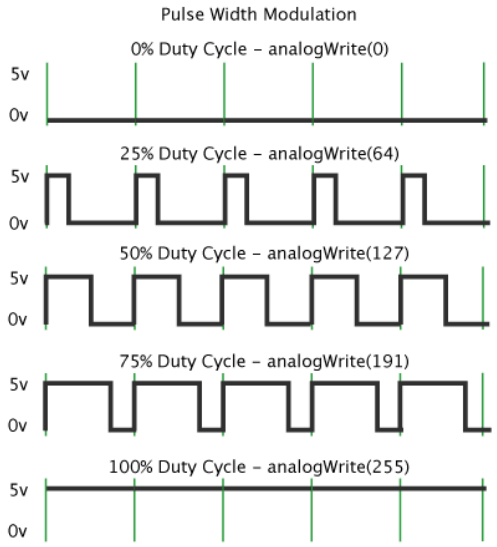

Figura 4.1 Señal PWM utilizada en radio control [32]

La señal PWM del receptor RC varía en un rango de [0, 2000] µs en función de la posición de los joysticks e interruptores del mando de radio control, que aumentarán o disminuirán el ciclo de trabajo y, por tanto, el valor digital equivalente.

Es posible transmitir y leer fácilmente este tipo de señales, aunque necesitaremos leer una señal PWM para cada uno de los canales del receptor RC.

Mediante la modulación por posición de pulso (PPM), es posible transmitir la misma información utilizando menos canales, sin embargo, presenta dos problemas fundamentales:

- Es necesario decodificar la información recibida.
- No todos los receptores de RC tienen salida PPM.

A continuación se muestra un ejemplo de señal PPM que contiene la información de tres señales PWM, que en este caso corresponden a servos.

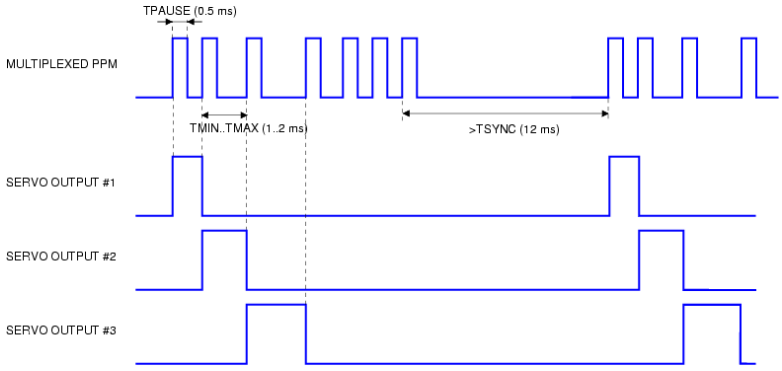

Figura 4.2 Señal PPM utilizada en radio control [33]

Los datos que recibimos del receptor de radio control se utilizarán para controlar el vuelo manual del cuadricóptero, por lo que es fundamental que sean precisos. Además debemos leerlos lo más rápido posible para optimizar nuestro código y disminuir el tiempo de ciclo entre un loop y el siguiente ya que este parámetro es crítico a la hora de realizar el control PID.

Con el fin de obtener los mejores datos, hemos realizado la lectura de los mismos mediante dos métodos distintos. El primero, utilizando la función pulseIn() de la biblioteca de Arduino, y el segundo mediante interrupciones. Los resultados obtenidos con cada uno de estos métodos se compararán para elegir el método más adecuado.

4.1.2.a Lectura utilizando la biblioteca

Este método es la forma más sencilla de realizar la lectura de los datos. Básicamente se trata de utilizar la función pulseIn() para leer la señal PWM recibida a través del pin del Arduino correspondiente.

La sintaxis de esta función es la siguiente: *pulseIn(pin, value)*. Se asigna un pin de entrada y se especifica si value= HIGH o LOW para leer los pulsos a nivel alto o bajo. De este modo, la función lee el pulso, detecta un flanco de subida (HIGH) y arranca un timer. La cuenta se detiene al detectar un flanco de bajada y la función devuelve el valor almacenado. De este modo, podemos leer los datos del receptor de radiocontrol mediante el procedimiento que se explica a continuación.

Algoritmo 4.1 Lectura de señal PWM utilizando la biblioteca

Algorithm 1: getDataRC

Input: Pines de entrada: $Roll\_IN\_Pin, Pitch\_IN\_Pin, Throttle\_IN\_Pin, Yaw\_IN\_Pin$
Output: Variables que almacenan los datos recibidos por RC: $Roll\_rx, Pitch\_rx, Throttle\_rx, Yaw\_rx$

```
1 begin
2   Roll_rx ← pulseIn(Roll_IN_Pin, HIGH);
3   Pitch_rx ← pulseIn(Pitch_IN_Pin, HIGH);
4   Throttle_rx ← pulseIn(Throttle_IN_Pin, HIGH);
5   Yaw_rx ← pulseIn(Yaw_IN_Pin, HIGH);
6 return
```

Sin embargo, la utilización de esta función implica la necesidad de recibir la información por polling, es decir, llamando a la función una vez cada loop, en vez de recoger la información cada vez que se actualiza el estado del pin.

4.1.2.b Lectura mediante interrupciones

La lectura mediante el uso de interrupciones nos permite recoger el valor de la señal del receptor RC únicamente cuando ésta actualice su valor, por lo que mientras tanto podemos estar dedicando nuestra atención y procesamiento a otras tareas más importantes.

La placa UDOO contiene un Arduino Due, el cual dispone de 76 pines disponibles y todos los pines digitales tienen la capacidad de generar una interrupción al cambiar de valor. Los dispositivos Arduino corrientes solo disponen de dos o cuatro pines capaces de generar interrupción.

Así pues, podemos crear una rutina de atención a la interrupción para recoger y almacenar el valor del pin. Para ello es necesario asignar la interrupción a dicho pin de la siguiente forma:

Algoritmo 4.2 Inicialización de las interrupciones

Algorithm 2: Interrrupt_initialize

Input: Pines asociados al receptor RC : $Roll\_IN\_Pin, Pitch\_IN\_Pin, Throttle\_IN\_Pin, Yaw\_IN\_Pin$
Output: Ninguna
Define Rutinas de atención a la interrupción : $ISR\_Roll, ISR\_Pitch, ISR\_Throttle, ISR\_Yaw$
begin
  $AttachInterrupt(Roll\_IN\_Pin, ISR\_Roll, CHANGE)$;
  $AttachInterrupt(Pitch\_IN\_Pin, ISR\_Pitch, CHANGE)$;
  $AttachInterrupt(Throttle\_IN\_Pin, ISR\_Throttle, CHANGE)$;
  $AttachInterrupt(Yaw\_IN\_Pin, ISR\_Yaw, CHANGE)$;
return

Por tanto, tenemos que especificar el pin, el nombre de la rutina de atención a la interrupción y si queremos que se active dicha interrupción en los flancos de subida y bajada (CHANGE), mientras que este activo (HIGH), únicamente en los flancos de subida (RISING), etc.

Para realizar la lectura y almacenado de los datos recibidos, es necesario utilizar la variable *bUpdateFlagsShared* que es un vector de flags, los cuales se activarán cada vez que se reciban nuevos valores. La asignación de los flags es la siguiente:

- *Roll_Flag:* 1_{10} → $0\,0\,0\,1_{\,2}$
- *Throttle_Flag:* 2_{10} → $0\,0\,1\,0_{\,2}$
- *Pitch_Flag:* 4_{10} → $0\,1\,0\,0_{\,2}$
- *Yaw_Flag:* 8_{10} → $1\,0\,0\,0_{\,2}$

A continuación hay que crear la rutina de atención a la interrupción (RAI o ISR por sus siglas en inglés). Se va a explicar únicamente una de ellas, puesto que todas ellas siguen el mismo proceso.

Algoritmo 4.3 Ejemplo de Rutina de Atención a la Interrupción

Algorithm 3: Rutina de atención a la interrupción. Ejemplo rutina ISR_Roll.

Input: Pin de entrada: $Roll\_In\_Pin$
Output: Variable global en la que se guarda el valor recibido: $Roll\_In\_Shared$,
1 **Define** Variable que almacena el tiempo en el que se activa la interrupción en microsegundos: $LastTime$.
Vector de flags: $bUpdateFlagShared$.
2 **begin**
3 **if** $digitalRead(Roll\_In\_Pin) = 1$ **then**
4 $LastTime \leftarrow micros()$;
else
5 $Roll\_In\_Shared \leftarrow micros() - LastTime$;
6 $bUpdateFlagShared \mid = Roll\_Flag$;
7 **return**

En esta RAI, si el pin está a nivel alto, se almacena el tiempo transcurrido, y si está a nivel bajo se almacena el valor en una variable global y se activa el flag correspondiente del *bUpdateFlagsShared*.

Algoritmo 4.4 Lectura de datos recogidos mediante el uso de interrupciones

Algorithm 4: Receiver_calculate

Input: Variables que contienen el último valor recibido por RC: $Roll\_In\_Shared$, $Pitch\_In\_Shared$, $Throttle\_In\_Shared$, Yaw_In_Shared.
Output: Variables locales que almacenan el valor de la señal recibida, con los que trabajaremos en cada ejecución del código: $Roll\_rx$, $Pitch\_rx$, $Throttle\_rx$, Yaw_rx.
1 **Define** Vector de flags: $bUpdateFlagsShared$. Flags para cada comando: $Roll\_flag$, $Pitch\_flag$, $Throttle\_flag$, Yaw_flag.
2 **begin**
3 **if** $bUpdateFlagsShared$ **then**
4 $noInterrupts()$;
5 **if** $bUpdateFlagsShared$ & $Roll\_flag$ **then**
6 $Roll\_rx \leftarrow Roll\_In\_Shared$
7 **if** $bUpdateFlagsShared$ & $Pitch\_flag$ **then**
8 $Pitch\_rx \leftarrow Pitch\_In\_Shared$
9 **if** $bUpdateFlagsShared$ & $Throttle\_flag$ **then**
10 $Throttle\_rx \leftarrow Throttle\_In\_Shared$
11 **if** $bUpdateFlagsShared$ & Yaw_flag **then**
12 $Yaw\_rx \leftarrow Yaw\_In\_Shared$
13 $interrupts()$;
14 $bUpdateFlagsShared = 0$;
15 **return**

En este algoritmo se comprueba *bUpdateFlagsShared*, desactivando las interrupciones para que se recojan los valores lo más rápidamente posible y se comprueban cada uno de los flags por separado, recogiendo el valor almacenado de cada uno de ellos en una variable local (Por ejemplo *Roll_rx*).

Roll_rx es una variable local que utilizaremos como setpoint o punto deseado en el controlador PID. Por último, se vuelven a activar las interrupciones y se limpian los flags de *bUpdateFlagsShared*.

4.1.3 Resultados obtenidos

Es importante recoger correctamente los datos del receptor de radio control puesto que si estos no se corresponden con la realidad, el quadcopter no tiene forma de saberlo. De modo que debemos asegurarnos de coger unos valores precisos, estables y realizarlo en el menor tiempo posible puesto que la frecuencia de reloj del Arduino integrado en la placa UDOO no es excesivamente alta.

Hemos realizado distintas pruebas para comprobar el funcionamiento de cada uno de los métodos de lectura de datos mencionados anteriormente. En estas pruebas se muestran los valores roll, pitch, yaw y throttle recibidos al realizar la siguiente secuencia:

1. Mantenemos el joystick en su posición de referencia
2. Movemos lentamente el joystick hasta su posición máxima
3. Mantenemos el joystick en la posición anterior
4. Movemos lentamente el joystick hasta su posición mínima

Sin embargo, para comentar las diferencias entre los valores recibidos, vamos a analizar los datos obtenidos con cada uno de los métodos realizando las pruebas de la secuencia anterior individualmente.

Tabla 4.1 Datos de RC. PulseIn vs Interrupciones (valores de referencia)

Valores de referencia (sin filtrar)							
PulseIn()				Interrupciones			
Roll	Pitch	Yaw	Throttle	Roll	Pitch	Yaw	Throttle
1433.00	1430.00	1390.00	1133.00	1514.00	1493.00	1508.00	1162.00
1361.00	1358.00	1421.00	1062.00	1514.00	1493.00	1507.00	1161.00
1385.00	1381.00	1381.00	1086.00	1513.00	1494.00	1509.00	1162.00
1420.00	1350.00	1379.00	1121.00	1513.00	1492.00	1507.00	1161.00
1450.00	1448.00	1408.00	1050.00	1513.00	1492.00	1509.00	1162.00
1478.00	1376.00	1439.00	1080.00	1514.00	1492.00	1509.00	1162.00
1403.00	1389.00	1357.00	1036.00	1514.00	1493.00	1508.00	1161.00
1439.00	1435.00	1396.00	1139.00	1514.00	1492.00	1508.00	1160.00
1366.00	1364.00	1427.00	1068.00	1513.00	1492.00	1508.00	1162.00
1390.00	1388.00	1392.00	1093.00	1513.00	1493.00	1509.00	1162.00
1360.00	1424.00	1360.00	1127.00	1514.00	1491.00	1508.00	1161.00
1456.00	1352.00	1414.00	1056.00	1513.00	1491.00	1509.00	1162.00
1383.00	1382.00	1444.00	1085.00	1513.00	1492.00	1510.00	1162.00
1351.00	1352.00	1367.00	1060.00	1514.00	1493.00	1508.00	1161.00

En la tabla superior se representan los valores recibidos mediante ambos métodos cuando mantenemos todos los sticks del mando de radiocontrol en su posición de referencia.

Como podemos observar, la función pulseIn() arroja valores de roll, pitch, throttle y yaw que distan mucho de ser constantes. Por otro lado, utilizando interrupciones podemos obtener valores considerablemente más estables.

Si comparamos cuantitativamente las 14 medidas obtenidas para cada uno de los métodos, podemos observar que:

- Utilizando pulseIn(), el punto de referencia de pitch varía entre los valores 1350 y 1448. Es decir, tenemos una variación de 98 unidades que hará necesario aplicar un filtrado a las medidas para obtener buenos resultados.
- Utilizando interrupciones, obtenemos valores que oscilan entre 1491 y 1493, es decir, el valor de referencia varía sólo 2 unidades.

Necesitamos saber los valores de referencia de cada una de las señales ya que son datos que utilizaremos, por ejemplo, en la función que realiza el armado o desarme de los motores. Así mismo, necesitaremos conocer los valores mínimos y máximos de cada uno de los comandos roll, pitch, etc. Por lo que es fundamental asegurar que los valores recibidos son estables y adecuados en todo momento.

A continuación mostramos los resultados obtenidos al realizar las transiciones de cada uno de los comandos correspondientes. Para facilitar la comparación y no incrementar demasiado el tamaño de las tablas de resultados, se ha incluido un retraso de 100 ms aumentando así el periodo de muestreo.

Tabla 4.2 Datos de RC. PulseIn vs Interrupciones (Transición Roll y Pitch)

Transición de Roll (sin filtrar)	
PulseIn()	Interrupciones
1409.00	1514.00
1482.00	1521.00
1519.00	1557.00
1484.00	1557.00
1486.00	1568.00
1526.00	1580.00
1457.00	1592.00
1458.00	1602.00
1537.00	1606.00
1576.00	1616.00
1510.00	1627.00
1560.00	1637.00
1600.00	1643.00
1632.00	1662.00
1534.00	1664.00
1615.00	1673.00
1664.00	1683.00
1655.00	1688.00
1661.00	1700.00
1716.00	1705.00
1687.00	1723.00
1719.00	1735.00
1779.00	1750.00
1708.00	1768.00
1759.00	1807.00
1766.00	1810.00
1798.00	1816.00
1726.00	1828.00
1700.00	1840.00
1784.00	1868.00
1714.00	1893.00
1717.00	1895.00
1773.00	1895.00
1804.00	1894.00

Transición de Pitch (sin filtrar)	
PulseIn()	Interrupciones
1430.00	1492.00
1359.00	1494.00
1388.00	1496.00
1394.00	1512.00
1462.00	1530.00
1507.00	1534.00
1404.00	1550.00
1500.00	1562.00
1453.00	1576.00
1495.00	1597.00
1453.00	1609.00
1481.00	1623.00
1529.00	1630.00
1532.00	1651.00
1624.00	1669.00
1572.00	1671.00
1719.00	1678.00
1572.00	1691.00
1607.00	1717.00
1654.00	1723.00
1656.00	1730.00
1725.00	1735.00
1652.00	1740.00
1682.00	1743.00
1679.00	1747.00
1641.00	1760.00
1771.00	1764.00
1670.00	1771.00
1625.00	1789.00
1660.00	1799.00
1689.00	1798.00
1693.00	1799.00
1646.00	1800.00
1659.00	1801.00

En las tablas superiores se muestran los resultados obtenidos al realizar una transición desde el punto de referencia de roll y pitch hasta sus posiciones máximas. Si nos centramos en la tabla superior izquierda, podemos observar que los datos presentan una clara tendencia positiva, es decir, se van incrementando como cabía esperar.

Sin embargo, mirando al detalle los datos obtenidos mediante el uso de pulseIn(), vemos que aunque los valores tienden a incrementarse, presentan oscilaciones y de vez en cuando éstos se decrementan momentáneamente. Este fenómeno se observa en los valores remarcados en rojo. Por el contrario, utilizando interrupciones obtenemos datos notablemente más estables, cuya tendencia es positiva y sin la presencia de datos atípicos.

Tabla 4.3 Datos de RC. PulseIn vs Interrupciones (Transición Yaw y Throttle)

Transición de Yaw (sin filtrar)		Transición de Throttle(sin filtrar)	
PulseIn()	Interrupciones	PulseIn()	Interrupciones
1447.00	1510.00	1072.00	1157.00
1356.00	1512.00	1057.00	1158.00
1405.00	1512.00	1090.00	1163.00
1435.00	1520.00	1137.00	1182.00
1358.00	1531.00	1129.00	1185.00
1393.00	1543.00	1122.00	1204.00
1423.00	1557.00	1163.00	1207.00
1452.00	1566.00	1118.00	1218.00
1377.00	1577.00	1155.00	1245.00
1412.00	1590.00	1186.00	1245.00
1462.00	1595.00	1234.00	1253.00
1400.00	1612.00	1160.00	1267.00
1417.00	1625.00	1209.00	1267.00
1500.00	1632.00	1279.00	1299.00
1539.00	1646.00	1313.00	1308.00
1454.00	1650.00	1251.00	1328.00
1526.00	1656.00	1298.00	1353.00
1579.00	1673.00	1346.00	1368.00
1507.00	1677.00	1290.00	1387.00
1576.00	1686.00	1341.00	1397.00
1619.00	1699.00	1372.00	1428.00
1559.00	1707.00	1399.00	1431.00
1564.00	1717.00	1357.00	1456.00
1651.00	1729.00	1424.00	1469.00
1699.00	1744.00	1542.00	1490.00
1616.00	1754.00	1434.00	1513.00
1711.00	1751.00	1468.00	1524.00
1773.00	1753.00	1523.00	1547.00
1727.00	1767.00	1483.00	1572.00
1723.00	1779.00	1535.00	1589.00
1787.00	1788.00	1586.00	1619.00
1818.00	1797.00	1652.00	1637.00
1715.00	1809.00	1587.00	1654.00
1722.00	1822.00	1647.00	1676.00
1805.00	1843.00	1706.00	1687.00
1833.00	1858.00	1630.00	1718.00
1714.00	1868.00	1663.00	1738.00
1792.00	1882.00	1693.00	1741.00
1823.00	1899.00	1723.00	1777.00
1751.00	1913.00	1680.00	1804.00
1749.00	1914.00	1709.00	1814.00
1814.00	1914.00	1725.00	1813.00

Como hemos podido observar, en las transiciones de los comandos roll, pitch, yaw y throttle se produce el mismo comportamiento en cuanto a tendencia positiva y presencia de datos atípicos cuando utilizamos la función pulseIn().

No obstante, los datos del comando throttle son ligeramente distintos puesto que el valor de referencia coincide con el valor mínimo. Lo cual es lógico, puesto que en un mando de radio control el stick de throttle siempre debe estar en su posición más baja. De lo contrario, los motores comenzarían a girar elevando el quadcopter.

Si representamos en una misma gráfica los resultados obtenidos utilizando ambos métodos de lectura de datos, para los ensayos de transiciones anteriormente mencionados, obtenemos las siguientes representaciones.

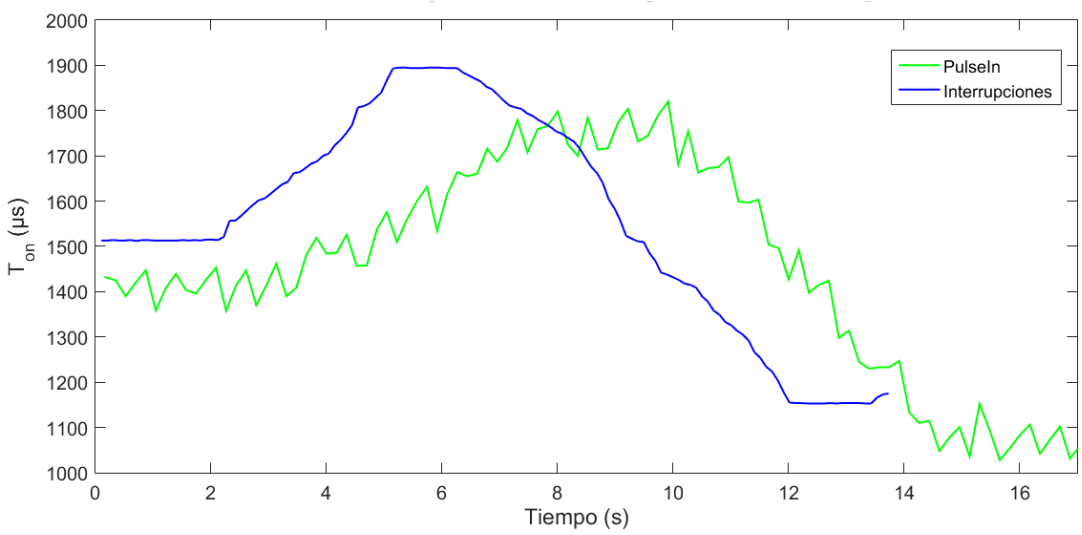

Figura 4.3 Lectura de datos RC. PulseIn vs Interrupciones (Transición de Roll)

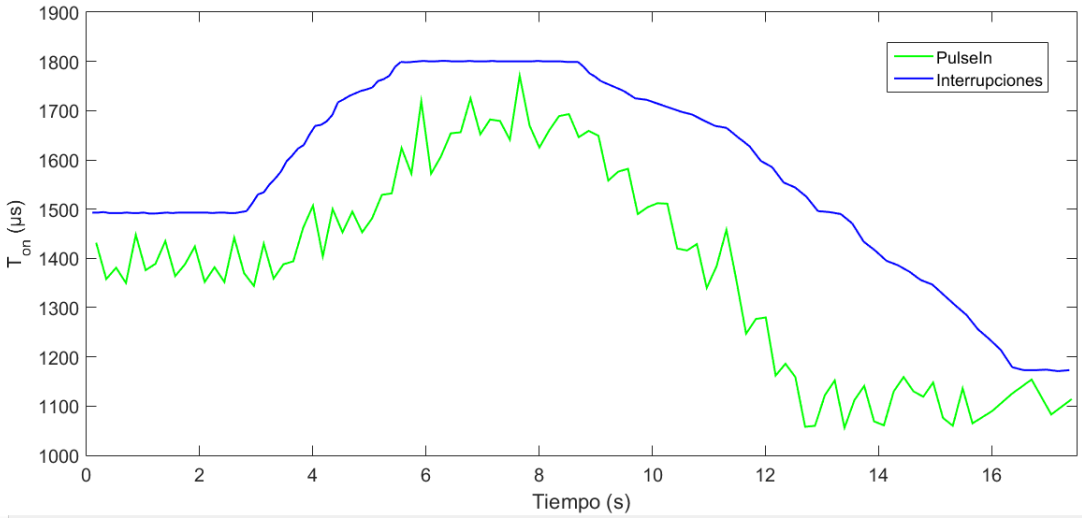

Figura 4.4 Lectura de datos RC. PulseIn vs Interrupciones (Transición de Pitch)

Cabe destacar que los ensayos se han realizado en momentos distintos por lo que pueden estar desfasados como en la "Figura 4.3", sin embargo, estas gráficas nos permiten comparar cualitativamente ambos métodos de lectura de datos.

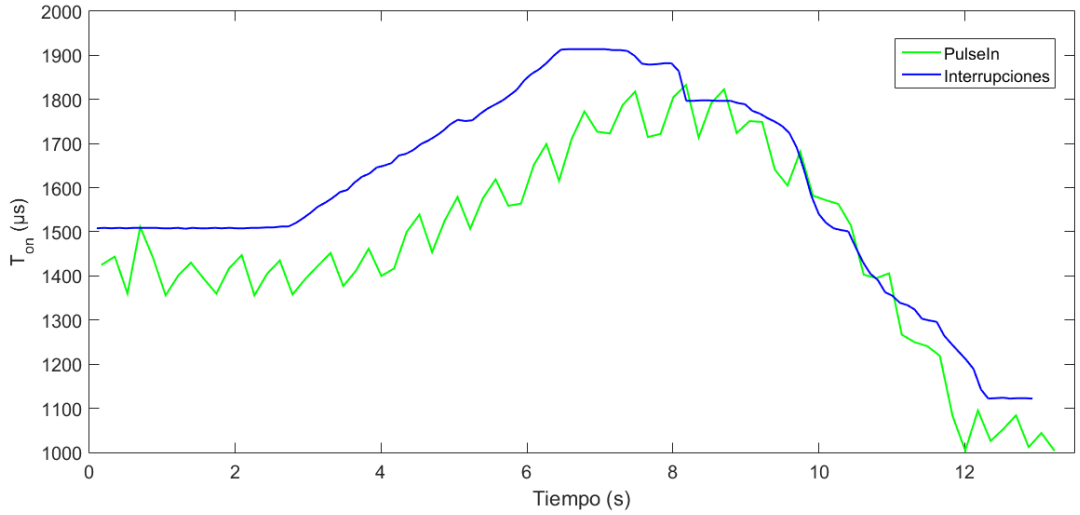

Figura 4.5 Lectura de datos RC. PulseIn vs Interrupciones (Transición de Yaw)

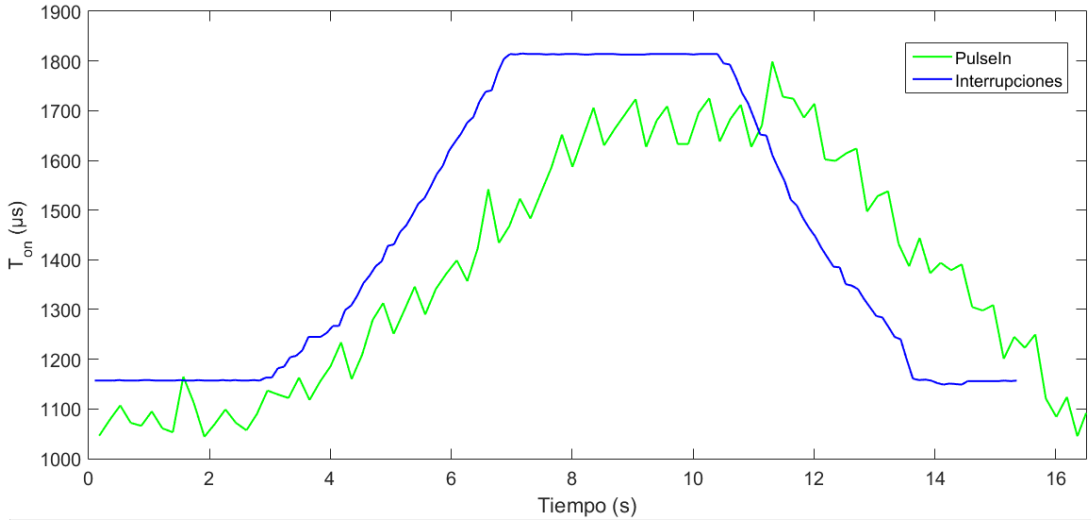

Figura 4.6 Lectura de datos RC. PulseIn vs Interrupciones (Transición de Throttle)

Atendiendo a las gráficas anteriores, podemos afirmar que los valores recibidos utilizando la librería de Arduino presentan un rizado bastante pronunciado por lo que, aunque filtrásemos la señal, no obtendríamos unos resultados tan buenos como los obtenidos mediante interrupciones.

Por último, para determinar la viabilidad de los métodos de lectura de datos, debemos comparar los tiempos de ejecución de cada uno de ellos. En este sentido, hemos realizado una prueba de lectura mientras que mantenemos los sticks del transmisor de radio control en su posición mínima, obteniendo los siguientes resultados:

Tabla 4.4 Tiempos de ejecución PulseIn vs Interrupciones

PulseIN Time		Interrups Time	
Time:	74.00	Time:	1.00
Time:	73.00	Time:	1.00
Time:	74.00	Time:	1.00
Time:	74.00	Time:	1.00
Time:	73.00	Time:	1.00
Time:	74.00	Time:	1.00
Time:	74.00	Time:	1.00
Time:	73.00	Time:	1.00
Time:	74.00	Time:	1.00
Time:	74.00	Time:	1.00
Time:	73.00	Time:	1.00
Time:	74.00	Time:	1.00
Time:	74.00	Time:	2.00
Time:	73.00	Time:	1.00

Obteniendo así una media de tiempos de ejecución de $\begin{cases} t_{PulseIn}=73,\ 64 \text{ ms} \\ t_{Interrupts}=1{,}07 \text{ ms} \end{cases}$

Por tanto, a la vista de los resultados, utilizaremos el método de lectura de datos mediante interrupciones puesto que arroja unos resultados considerablemente estables, con una variación máxima de 3 unidades respecto a su valor de referencia (lo cual representa una desviación de 0.3%) y con un tiempo de ejecución de 1 ms, es decir, casi 70 veces más pequeño.

4.2 Unidad de medición inercial

4.2.1 Conexionado

La unidad de medición inercial (IMU) elegida es *Adafruit 10-DOF IMU Breakout* que dispone de tres giróscopos, tres acelerómetros y tres magnetómetros, los cuales se comunican con el Arduino mediante la comunicación I2C.

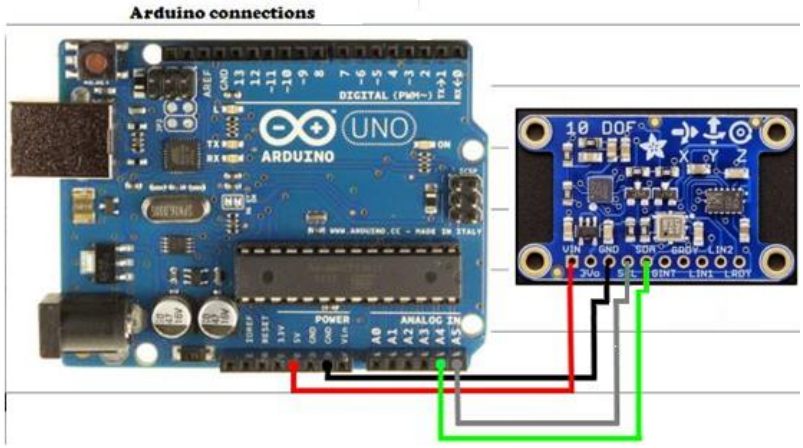

Figura 4.7 Conexionado de Adafruit 10 DOF IMU [34],[35]

En este tipo de comunicaciones la información se transmite a través de dos cables conectados a los pines SCL y SDA. Por tanto es necesario conectar los pines SDA y SCL de la IMU a los pines correspondientes en la placa Arduino. También es necesario conectar los pines VIN y GND. Las conexiones se han de realizar como se ilustra en la imagen superior.

4.2.2 Driver y estimación del ángulo

Ahora que tenemos todo conectado, procedemos a implementar el código necesario para obtener la información de cada uno de los sensores de la IMU. Para ello, el fabricante Adafruit pone a nuestra disposición una serie de librerías que manejan la comunicación a bajo nivel entre el Arduino y el dispositivo.

Por tanto, debemos incluir las librerías como se muestra a continuación.

- *Adafruit_10DOF*
- *Adafruit_Sensor*
- *Adafruit_L3GD20_U*
- *Adafruit_LSM303_U*
- *Adafruit_BMP085_U*

Estas librerías se pueden encontrar en la página oficial de Adafruit o a través de GitHub.

En primer lugar, es necesario declarar una serie de objetos de las clases que se muestran a continuación e inicializarlas llamando al constructor correspondiente. De lo contrario, no seremos capaces recibir datos de la unidad de medición inercial.

```
Adafruit_10DOF    dof   = Adafruit_10DOF();
Adafruit_LSM303_Accel_Unified   accel= Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified     mag   = Adafruit_LSM303_Mag_Unified(30302);
Adafruit_L3GD20_Unified         gyro  = Adafruit_L3GD20_Unified(20);
sensors_vec_t   orientation;
```

La clase *sensors_vect_t* tiene como atributos roll, pitch y heading, por lo que en el objeto *orientation* estarán contenidos los ángulos de Euler.

Con el fin de ilustrar el funcionamiento del driver de la IMU, procederemos a explicar los métodos de obtención de las medidas. Los cuales se han dividido en pequeños y sencillos algoritmos. Para obtener los valores de los giróscopos, utilizaremos el procedimiento descrito en el siguiente algoritmo:

Algoritmo 4.5 Recogida de datos del giróscopo

Algorithm 5: Gyro_calculate

Input: Ninguna
Output: Velocidad angular expresada en grados por segundo: $gyro\_roll, gyro\_pitch, gyro\_yaw$.
1 **Define** *Sensors_event_t* es una clase contenida en la librería de Adafruit. El offset del giróscopo obtenido realizando la calibración del sensor: $gyro\_roll\_offset, gyro\_pitch\_offset, gyro\_yaw\_offset$
2 **begin**
3 $sensors\_event\_t\ event$;
4 $gyro.getEvent(\&\ event)$;
5 $gyro\_roll \leftarrow (event.gyro.x) \cdot 180/\pi\ + gyro\_roll\_offset$;
6 $gyro\_pitch \leftarrow (event.gyro.y) \cdot 180/\pi\ + gyro\_pitch\_offset$;
7 $gyro\_yaw \leftarrow (event.gyro.z) \cdot 180/\pi\ + gyro\_yaw\_offset$;
8 **return**

En primer lugar, se llama al método *getEvent()* para captar el evento del giróscopo, es decir, obtener las velocidades angulares en cada uno de los ejes en radianes por segundo. Después, realizamos la conversión a grados y realizamos un pequeño ajuste del sensor.

Ahora obtenemos los valores de los acelerómetros y de los magnetómetros:

Algoritmo 4.6 Recogida de datos del acelerómetro y magnetómetro

Algorithm 6: Acc_calculate

Input: Ninguna
Output: Objeto que contiene los ángulos de euler obtenidos utilizando los datos del acelerómetro: *orientation*
1 **Define** *Sensors_event_t* es una clase contenida en la librería de Adafruit.
2 **begin**
3 $sensors\_event\_t\ accel\_event$;
4 $sensors\_event\_t\ mag\_event$;
5 $accel.getEvent(\&\ accel\_event)$;
6 $mag.getEvent(\&\ mag\_event)$;
7 $orientation \leftarrow dof.accelGetOrientation(\&accel\_event)$;
8 **return**

Al igual que con los giróscopos, captamos el evento para obtener los valores y llamamos a la función *accelGetOrientation()*, que convierte los valores transmitidos por el acelerómetro (radianes por segundo al cuadrado) en ángulos de orientación (expresado en grados).

Por último, implementamos un filtro complementario para eliminar el ruido a bajas frecuencias del giróscopo y las perturbaciones a altas frecuencias del acelerómetro, obteniendo así unos ángulos de Euler mucho más precisos.

Algoritmo 4.7 Calculo de los ángulos de Euler utilizando un filtro complementario

Algorithm 7: imu_update

Input: Velocidad angular expresada en grados por segundo: $gyro\_roll, gyro\_pitch, gyro\_yaw$. Orientación obtenida utilizando los datos del acelerómetro: $orientation$.

Output: Los ángulos de Euler resultantes se almacenan en: $Roll\_imu$, $Pitch\_imu$ y Yaw_imu.

1 **Define** Utilizamos los algoritmos $gyro\_calculate$ y $acc\_calculate$ ya que vamos a fusionar ambas medidas utilizando un filtro complementario. Tiempo actual en milisegundos: now, tiempo de la anterior ejecución en milisegundos: $last$, tiempo transcurrido en segundos: dt.

2 **begin**

3 **if** $acc\_calculate()$ **then**

4 $gyro\_calculate()$;

5 $now \leftarrow millis()$;

6 $dt \leftarrow (now - last)/1000$;

7 $Roll\_imu \leftarrow 0.98 \cdot (Roll\_imu + gyro\_roll \cdot dt) + 0.02 \cdot orientation.roll)$;

8 $Pitch\_imu \leftarrow 0.98 \cdot (Pitch\_imu + gyro\_pitch \cdot dt) + 0.02 \cdot orientation.pitch)$;

9 $Yaw\_imu \leftarrow (Yaw\_imu + gyro\_yaw \cdot dt)$;

10 $last \leftarrow now$;

11 **return**

Como podemos observar, no se ha utilizado el acelerómetro para calcular el ángulo de rotación sobre el eje z (yaw) puesto que éste es ortogonal a la gravedad, lo que provoca que las medidas obtenidas con el acelerómetro no se correspondan con la realidad.

4.2.3 Calibración del offset de la IMU

La unidad de medición inercial, como todo dispositivo electrónico, necesita ser calibrado para que funcione correctamente. Los sensores giroscópicos son lo más susceptibles a errores de calibración por lo que es fundamental que ésta se realice correctamente.

Para ello hemos utilizado el siguiente método de calibración, el cual toma 1000 muestras de la velocidad angular y hace la media de todas ellas. De este modo, obtenemos la velocidad angular media cuando la IMU se encuentra parada sobre una superficie estable, lo cual es fundamental para que la calibración resulte eficaz.

Algoritmo 4.8 Calibración de los sensores giroscópicos

Algorithm 8: Gyro_calibration

Input: Ninguna
Output: Los valores del offset resultante se almacenan en $Roll\_calibrate$, $Pitch\_calibrate$ y $Yaw\_calibrate$.

```
Define  Utilizamos el algoritmo gyro_calculate para realizar la calibración del sensor.
begin
for i=0 ; i < 1000 ; i++ do
    gyro_calculate();
    roll_calibrate ← roll_calibrate + gyro_roll;
    pitch_calibrate ← pitch_calibrate + gyro_pitch;
    yaw_calibrate ← yaw_calibrate + gyro_yaw;
roll_calibrate ← roll_calibrate/1000;
pitch_calibrate ← pitch_calibrate/1000;
yaw_calibrate ← yaw_calibrate/1000;
return
```

Después de realizar el proceso de calibración en la Adafruit 10 DOF IMU, hemos obtenido los siguientes resultados:

- $Gyro\_roll\_offset$ = 0.25473874995422649547 °/seg
- $Gyro\_pitch\_offset$ = -1.83176000306538515616 °/seg
- $Gyro\_yaw\_offset$ = -1.57966374316363395990 °/seg

Por último, bastará con restar el offset calculado a las medidas de velocidad angular provenientes de los giróscopos. Lo cual nos permitirá obtener los ángulos de Euler precisos simplemente integrando la velocidad angular rectificada.

4.2.4 Resultados obtenidos

4.2.4.a Gyro vs Acc vs Comp Filter (Adafruit IMU)

Con el fin de comprobar el correcto funcionamiento de la unidad de medición inercial *Adafruit 10 DOF IMU*, se van a representar los ángulos de Euler (roll, pitch y yaw) rotando la IMU unos ángulos fijos.

Para ello representaremos tres curvas en una misma gráfica, en la primera se muestran los ángulos obtenidos utilizando únicamente los giróscopos, en la segunda lo ángulos obtenidos con los acelerómetros y en la última los ángulos resultantes de utilizar un filtro complementario sobre las medidas de los sensores anteriormente mencionados.

En primer lugar, se representa el ángulo pitch si mantenemos la IMU estable sobre una superficie plana con una referencia de 0 grados.

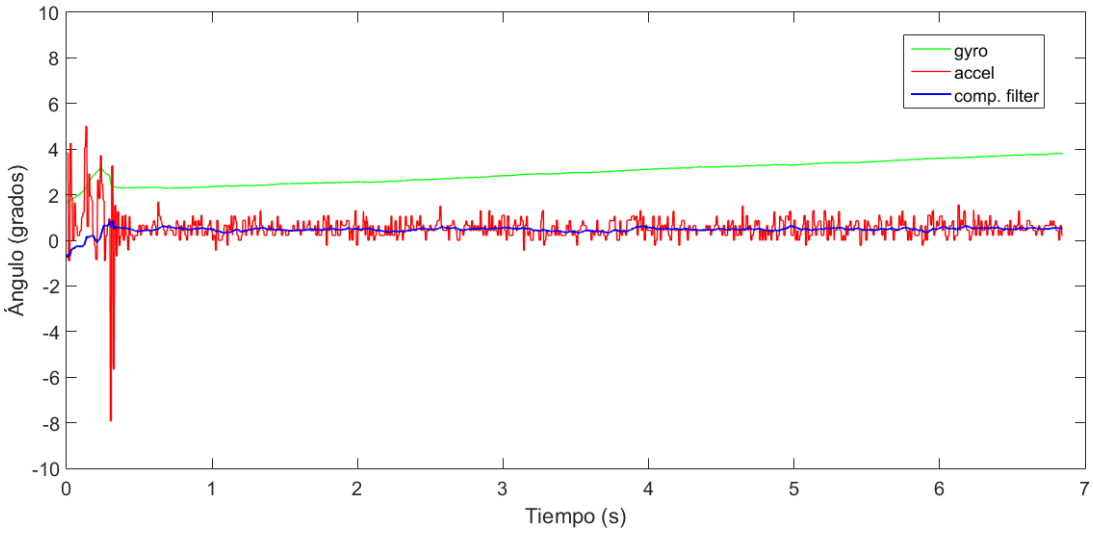

Figura 4.8 Ángulo Pitch en Adafruit 10 DOF IMU con referencia fija de 0 grados

En esta imagen se aprecia claramente las ventajas y defectos de los sensores utilizados. El giróscopo da una medida de ángulo muy estable que, sin embargo, tiende a crear una deriva de 0.3 °/s aproximadamente que va incrementando el ángulo poco a poco. Por otro lado, el acelerómetro da un ángulo más exacto aunque presenta un rizado a altas frecuencias.

Al utilizar el filtro complementario, filtramos el rizado del acelerómetro y compensamos el "drift" del giróscopo. De este modo, obtenemos un ángulo de pitch constante cuyo valor es de 0.2 °.

Ahora que hemos comprobado el correcto funcionamiento de la IMU en una superficie estable, es el momento de examinar su comportamiento al realizar un movimiento desde 0 grados hasta -45 grados.

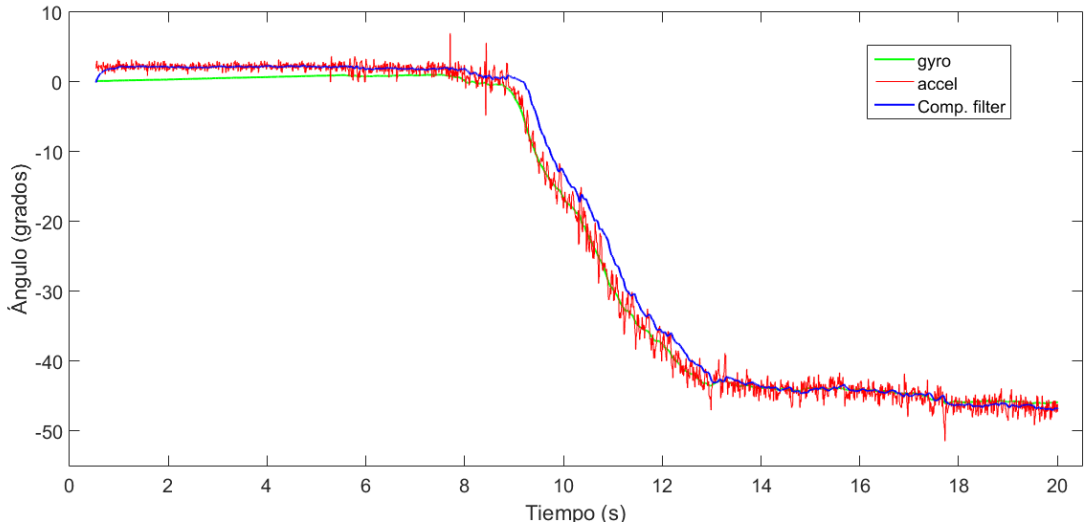

Figura 4.9 Ángulo Pitch en Adafruit 10 DOF IMU (transición de 0 a -45 grados)

Como podemos observar, el ángulo pitch se va decrementando paulatinamente hasta alcanzar la referencia deseada, es decir, el ángulo de -45 grados. La curva correspondiente al filtro complementario sigue perfectamente al giróscopo y ajusta la deriva con los datos procedentes del acelerómetro.

Al realizar el mismo ensayo con una transición de 0 a 45 grados, obtenemos la siguiente gráfica.

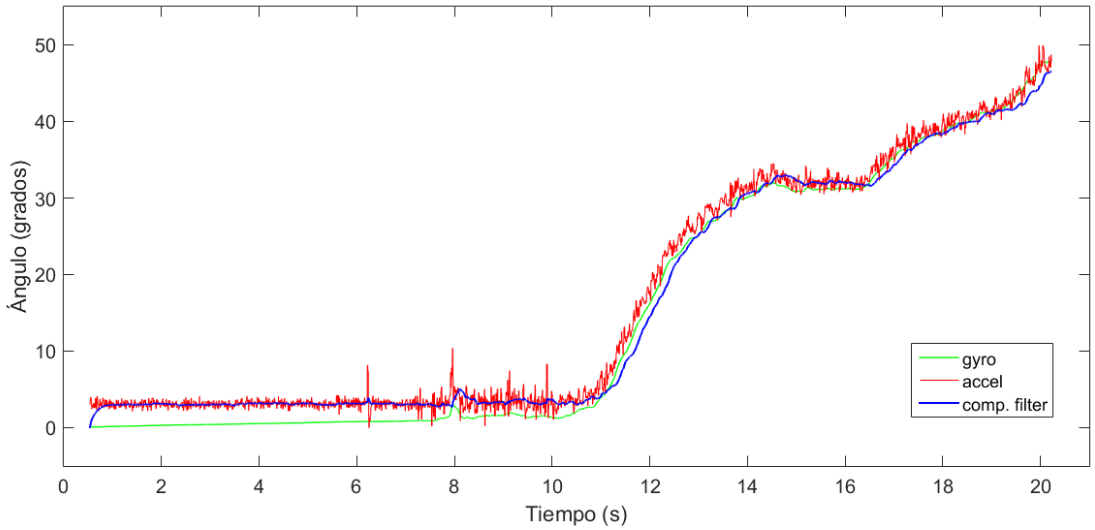

Figura 4.10 Ángulo Pitch en Adafruit 10 DOF IMU (transición de 0 a 45 grados)

Cabe destacar que el banco de pruebas utilizado es algo rudimentario y el movimiento de la unidad de medición inercial se ha realizado manualmente.

Por tanto, en algunas gráficas se han producido ciertas fluctuaciones en la transición del ángulo inicial al de referencia. Lo cual no se debe de considerar como un fallo de la IMU sino como fallo inherente a la realización manual de dichas pruebas.

Si nos fijamos en las gráficas mostradas anteriormente, podemos observar que éstas corresponden al ángulo de cabeceo o pitch. No se ha incluido las gráficas de alebeo o roll para no extendernos demasiado, ya que éstas mostrarían la misma respuesta que la explicada anteriormente.

En cambio sí que vamos a mostrar el ángulo yaw, puesto que es ortogonal a la gravedad y, por tanto, el acelerómetro no funcionará correctamente. En un principio, para solucionar este problema, se propuso utilizar los datos obtenidos con los magnetómetros para fusionarlos con los del giróscopo, obteniendo así un ángulo más preciso.

Sin embargo, después de realizar distintas pruebas hemos llegado a la conclusión de que los valores arrojados por el magnetómetro para el ángulo yaw no se corresponden con la realidad, puesto que tienen una deriva mucho mayor que el giróscopo. De modo que, finalmente, se ha utilizado únicamente el valor obtenido por los giróscopos.

La grafica siguiente muestra el ángulo yaw para una referencia fija de 0 grados.

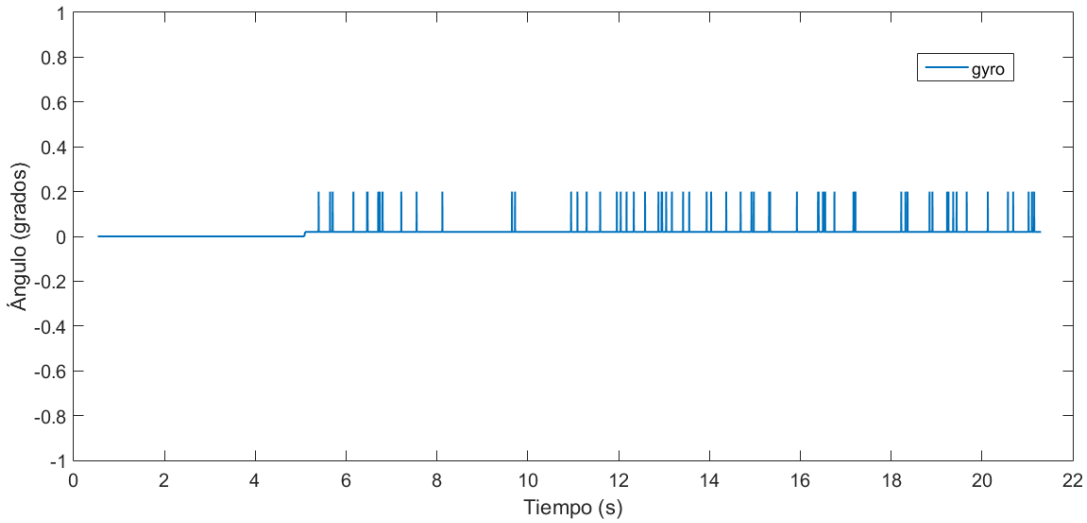

Figura 4.11 Ángulo Yaw en Adafruit 10 DOF IMU con referencia fija de 0 grados

Los resultados de la gráfica superior muestran que el giróscopo aporta unos valores notablemente estables, cuyo valor varía entre los 0 grados y los 0.2 grados.

Esto es así puesto que para evitar la aparición del "drift" se ha filtrado la señal de modo que para variaciones inferiores 1 °/seg, la señal permanezca constante.

Al igual que en los ensayos del ángulo pitch, hemos realizado una prueba en la que se muestran los valores obtenidos al efectuar una transición entre 0 y 90 grados.

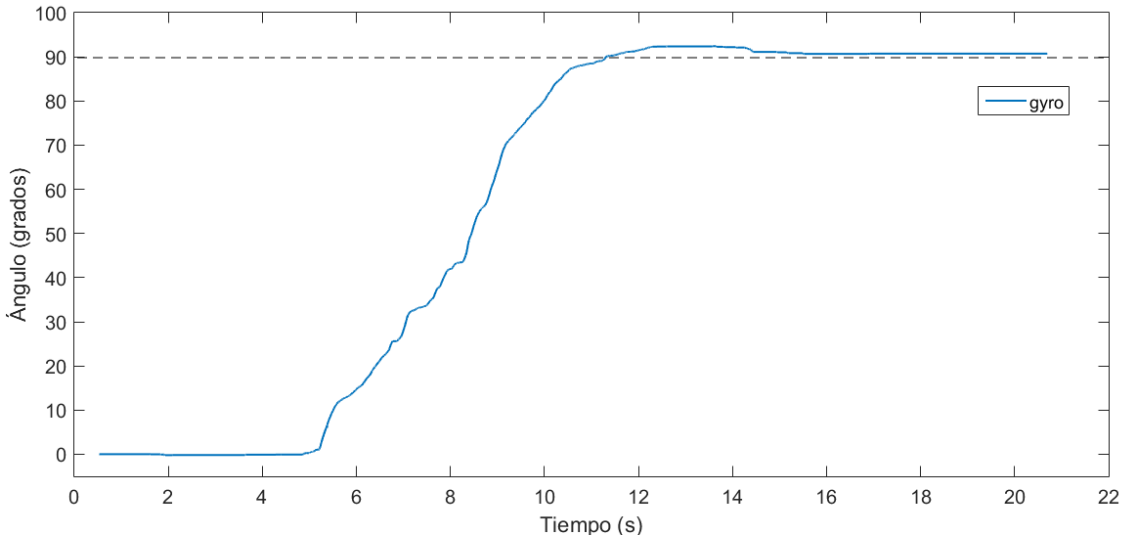

Figura 4.12 Ángulo Yaw en Adafruit 10 DOF IMU (transición de 0 a 90 grados)

Teniendo en cuenta que únicamente estamos utilizando un sensor para la obtención de los resultados, éstos son considerablemente fieles a la realidad. Realizando ensayo anterior para una transición en el sentido contrario obtenemos la siguiente gráfica.

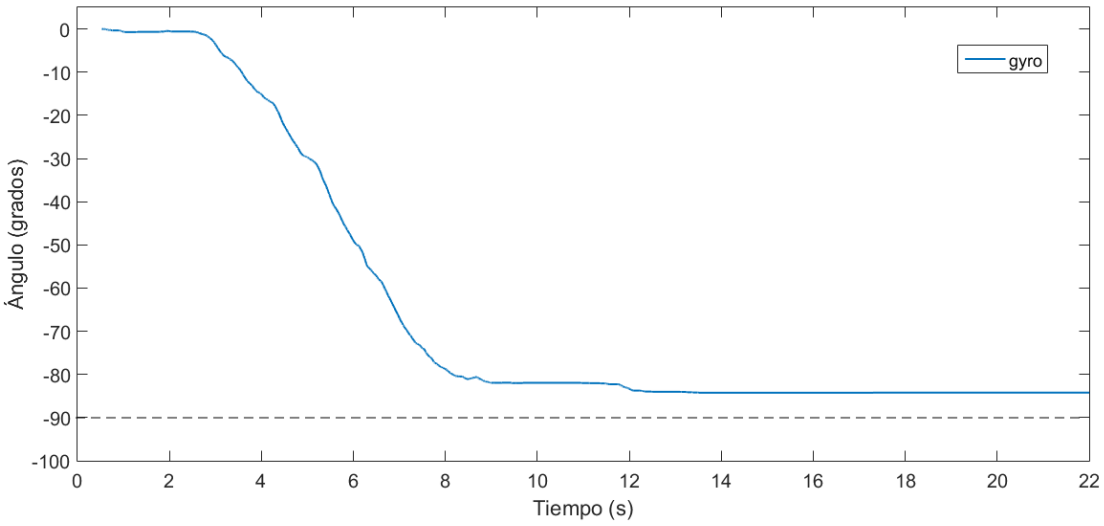

Figura 4.13 Ángulo Yaw en Adafruit 10 DOF IMU (transición de 0 a -90 grados)

En esta prueba se aprecia claramente que se ha producido una deriva de aproximadamente 6 grados. Por lo tanto, es preciso realizar un filtrado mayor que nos permita limitar este efecto.

En definitiva, a la vista de las pruebas realizadas, podemos determinar que la unidad de medición inercial seleccionada proporciona ángulos de Euler muy precisos gracias a la fusión sensorial de los sensores que incorpora en un mismo dispositivo electrónico.

4.2.4.b Precisión Adafruit IMU vs Hobbyking Control Board IMU

Cuando afirmamos que la *Adafruit IMU 10 DOF* es una unidad inercial potente que arroja unos valores precisos de los ángulos de Euler, no es una opinión personal, sino que está basado en las pruebas realizadas y en la comparación con otros dispositivos inerciales similares.

En este sentido, hemos utilizado la unidad inercial contenida en la placa controladora *Hobbyking Control board* para obtener los ángulos roll, pitch y yaw durante los ensayos realizados anteriormente. Ésta placa controladora obtiene los ángulos utilizando tres sensores giroscópicos que contiene.

A continuación se muestra el ángulo pitch al mantener la IMU en un ángulo de referencia de 0 grados para cada una de las unidades inerciales.

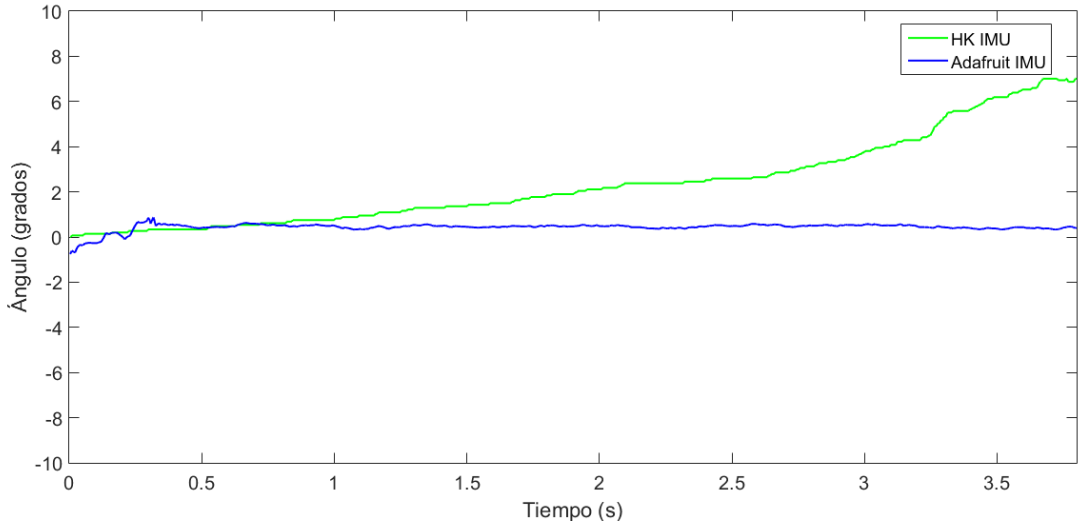

Figura 4.14 HK IMU vs Adafruit 10 DOF IMU (Referencia fija en 0 grados)

Como podemos observar, la *Hobbyking Control Board IMU* arroja unos ángulos que presentan una deriva de 1.33°/seg que no puede ser compensada puesto que dicha placa solamente tiene giróscopos. Por el contrario, el valor de referencia para la *Adafruit IMU* se mantiene en un valor constante de 0.2°.

En las dos siguientes gráficas se muestra el ángulo pitch obtenidos al realizar una transición de 0 a 45 grados y otra de 0 a -45 grados.

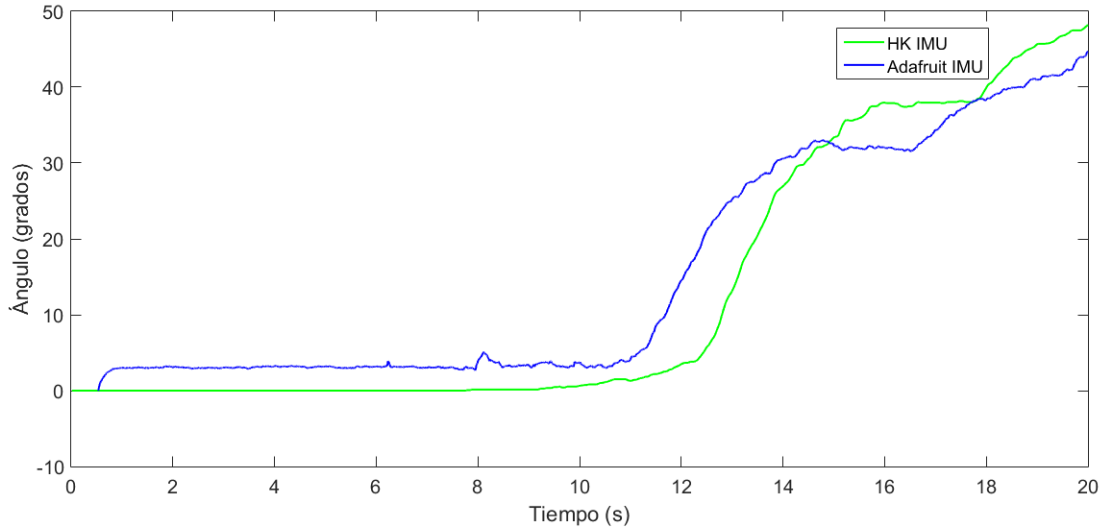

Figura 4.15 HK IMU vs Adafruit 10 DOF IMU (Transición de 0 a 45 grados)

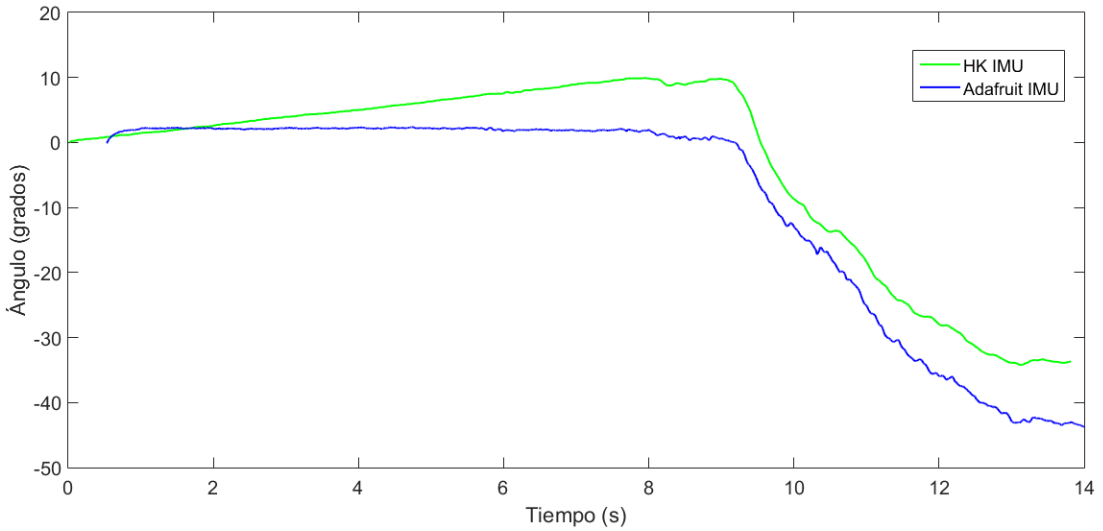

Figura 4.16 HK IMU vs Adafruit 10 DOF IMU (Transición de 0 a -45 grados)

En éstas gráficas se aprecia claramente la deriva de la medida de ángulo obtenido utilizando la *Hobbyking Control Board IMU*. Dicha deriva provoca que en lugar de alcanzar el ángulo de referencia de 45° se obtiene una medida de casi 50°. En el caso de la transición desde 0 hasta -45 grados, solo alcanzamos los -32°, esto es debido a la deriva de aproximadamente 10 grados que se produce en los primeros instantes.

En cambio, la *Adafruit IMU* proporciona unos valores considerablemente más estables y en los que la deriva de los giróscopos no sólo es menor, sino que ésta se compensa gracias a la presencia de acelerómetros.

4.2.4.c Tiempo de ejecución vs Tiempo loop

Cuando controlamos un sistema en tiempo real, el tiempo de muestreo es un parámetro crítico, puesto que influye directamente en el sistema de control y por tanto en el comportamiento del sistema.

En este sentido, es prioritario minimizar los tiempos de ejecución, lo que nos ha llevado a comprobar qué partes del código requieren más tiempo. El tiempo transcurrido entre un bucle y el siguiente es denominado tiempo de loop, que en nuestro caso es de 2.8 ms.

Después de realizar las comprobaciones necesarias, hemos llegado a la conclusión de que las funciones *Serial.print()*, *Serial.println()* y todas las funciones que impliquen imprimir por pantalla, precisan de tiempos del orden de ms, lo que comparándolo con el tiempo de loop representan un porcentaje muy significativo del tiempo total. Este tipo de funciones son imprescindibles para comprobar el correcto funcionamiento del código pero han de ser eliminadas en el momento en que trabajemos en tiempo real.

Con respecto al resto de funciones, las que necesitan un mayor tiempo para ejecutarse se resumen en la siguiente tabla.

Tabla 4.5 Tiempos de lectura de giróscopo, acelerómetro y magnetómetro

Nombre de la función	Tiempo de ejecución
gyro.getEvent(&event);	0.884 ms
accel.getEvent(&accel_event);	0.87 ms
mag.getEvent(&mag_event);	1.48 ms

Como es lógico, las funciones que demandan un mayor esfuerzo en cálculos aritméticos son las que se ejecutan más lentamente. Todas ellas están relacionadas con la IMU. Como hemos explicado anteriormente, los valores obtenidos con el magnetómetro son muy erráticos. Esto unido con que el tiempo de ejecución del sensor es notablemente superior al del giróscopo y acelerómetro, nos ha llevado a excluir el magnetómetro de la estimación del estado.

4.3 Driver de los ESC

4.3.1 Principios de funcionamiento y armado de los motores

Un ESC es un controlador electrónico de velocidad que se utiliza para realizar el control de los motores brushless. Esto se realiza a través de tres cables, a saber, alimentación, tierra y señal. Como hemos comentado anteriormente, el variador dispone además de una alimentación externa, la cual entrega la tensión demandada por el dispositivo electrónico. Por tanto, del conjunto de tres cables debemos dejar en circuito abierto el correspondiente a la alimentación, de no ser así, el variador no funcionará correctamente.

Los ESC se controlan mediante una señal PWM, la cual generamos en un pin del Arduino. En este sentido, utilizaremos la librería Servo que nos permitirá controlar los variadores de la misma forma que controlaríamos un simple servo.

Los motores varían su velocidad dependiendo del tiempo que la señal se encuentra a nivel alto (T_{on}), el cual puede variar entre 1000 µs y 2000 µs, con un periodo de 20 ms. Cualquier PWM cuyo T_{on} esté fuera del rango anterior no será recibida correctamente por los ESC.

Puesto que el periodo de la señal generada es de 20 ms, el control de los motores se realiza a una frecuencia constante de 50 Hz. Lo cual coincide con la frecuencia de operación de la mayoría de los variadores electrónicos del mercado. Sin embargo, algunos ESC permiten aumentar la frecuencia de operación por lo que para controlarlos sería necesario disminuir el periodo de la señal PWM utilizada.

Para generar la señal PWM en el Arduino, bastará con declarar un objeto de la clase Servo, adjuntarlo al pin digital deseado y mandarle la señal de control deseada.

```
Servo servoOne;
Servo servoTwo;
Servo servoThree;
Servo servoFour;
```

Antes de continuar con el control de los motores, es imprescindible armarlos, puesto que si no lo hacemos éstos no se moverán. Para ello hay que enviar un pulso de 1000 µs a cada motor durante al menos cuatro segundos.

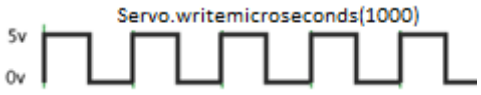

Figura 4.17 Señal de armado de motores

Esta señal se corresponde con un ángulo de 0 grados si utilizamos *servo.write()* en lugar de *servo.writemicroseconds()*. A continuación mostramos el proceso de armado de los motores.

Algoritmo 4.9 Proceso de armado de los motores

Algorithm 9: Motors_initialize

Input: Ninguna
Output: Ninguna
Define Los pines de salida a través de los cuales se controlan los motores son: $M1\_Out\_Pin$, $M2\_Out\_Pin$, $M3\_Out\_Pin$, $M4\_Out\_Pin$.
begin
  $servoOne.attach(M1\_Out\_Pin)$;
  $servoTwo.attach(M2\_Out\_Pin)$;
  $servoThree.attach(M3\_Out\_Pin)$;
  $servoFour.attach(M4\_Out\_Pin)$;
for $i=1 ; i \leq 4 ; i$++ **do**
  $Motor[i] \leftarrow 1000\mu s$;
$Wait\ 4\ seg$;
return

Cabe destacar que para informar de su estado, los motores brushless emiten una secuencia de pitidos que nos permiten distinguir cuando están armados y cuando no. Si hemos inicializado correctamente los motores, pasaremos de oír una secuencia de pitidos continuos, a una melodía que indica que están listos para funcionar.

Para armar los motores a través del transmisor de radio control, es necesario fijar una secuencia de movimiento de los sticks concreta y, de este modo, evitar que el quadcopter se ponga en marcha sin querer o que se desarme en pleno vuelo.

Para implementar esta funcionalidad hemos decidido que es necesario mantener el stick que controla el throttle al mínimo a la vez que el stick del pitch al máximo durante al menos tres segundos. Mientras que no se realice la secuencia de armado correctamente el quadcopter no funcionará, sin embargo debemos seguir enviando la señal PWM a los motores para que éstos no se desarmen.

Algoritmo 4.10 Armado del UAV

Algorithm 10: Armado del UAV

Input: Las variables $Pitch\_rx$ y $Throttle\_rx$ son comandos recibidos por radio control.
Output: Ninguna
Define El método $Receiver\_calculate()$ recoge los valores enviados por RC. $Pitch\_max$ es el valor máximo y $Throttle\_ref$ es su valor de referencia. Tiempo actual en milisegundos: t.
begin
while *armado=FALSE* **do**
  $Receiver\_calculate()$;
  if $Pitch\_rx > Pitch\_max\ \&\ Throttle\_rx < Throttle\_ref$ **then**
    $t \leftarrow millis()$;
    if $t > 3seg$ **then**
      $armado = TRUE$;
  else
    $t \leftarrow 0$;
  for $i=1 ; i \leq 4 ; i$++ **do**
    $Motor[i] \leftarrow acum\_M[i]\ \mu s$;
return

4.3.2 Control de los motores

El control de los motores está directamente relacionado con la configuración del cuadricóptero, es decir, los valores de las señales que debemos mandar a cada motor dependen de si utilizamos una configuración en + o una configuración en X.

Por tanto es fundamental tener presente los fundamentos de vuelo de cada una de estas configuraciones, de lo contrario, podríamos mandar la señal equivocada y el UAV acabaría estrellándose. (Para cualquier duda, consultar el apartado "Fundamentos de vuelo" incluido en el Estado del arte).

Durante el vuelo se realizan distintos movimientos al mismo tiempo, por ejemplo, para desplazarnos hacia delante a la izquierda, estamos realizando el movimiento de throttle (para que el dron se mantenga en el aire), el movimiento de pitch (para que se incline hacia delante) y el movimiento de roll (Para que se incline hacia la izquierda). Por tanto, la señal de los motores tendrá un valor asociado a cada uno de esos movimientos y no todos los motores están asociados a los mismos movimientos.

Para ilustrar éste fenómeno, procedemos a explicar brevemente el algoritmo que realiza esta funcionalidad.

Algoritmo 4.11 Proceso de control de los motores

Algorithm 11: Control_update

Input: La variable $Throttle\_rx$ es el comando enviado por RC. $Pitch\_out$, $Roll\_out$ y Yaw_out son los valores obtenidos como salida del regulador PID.
Output: Ninguna
Define Las variables $acum\_M1$, $acum\_M2$, $acum\_M3$ y $acum\_M4$ guardan el valor de la suma de cada una de sus componentes.
begin
  $acum\_M1 \leftarrow Throttle\_rx - Pitch\_out + Roll\_out + Yaw\_out$;
  $acum\_M2 \leftarrow Throttle\_rx - Pitch\_out - Roll\_out - Yaw\_out$;
  $acum\_M3 \leftarrow Throttle\_rx + Pitch\_out - Roll\_out + Yaw\_out$;
  $acum\_M4 \leftarrow Throttle\_rx + Pitch\_out + Roll\_out - Yaw\_out$;
for $i=1$; $i \leq 4$; i++ **do**
  $Motor[i] \leftarrow acum\_M[i]\ \mu s$;
return

Como podemos observar, el valor total acumulado del motor M1 se compone de la suma de los valores de throttle, roll, yaw y de la resta del valor del pitch. Podría parecer un poco confuso, pero resulta lógico pensar que el motor M1 irá más rápido cuando el quadcopter ascienda, vaya hacia la derecha o rote en sentido de las agujas del reloj y girará más lento el motor cuando el quadcopter se desplace hacia delante.

Por otro lado, aunque los motores se arman cuando reciben una señal PWM cuyo T_{on}=1000 µs, no empiezan a moverse hasta que alcanzan los 1100 µs por lo que si enviamos a los motores señales con un valor inferior a éste, el motor dejará de girar. El

tiempo que se pierde entre que el motor deja de girar y vuelve a arrancar, por muy pequeño que pueda parecer, afecta al control de vuelo y hace el quadcopter inestable.

Algo parecido ocurre cuando llegamos al límite máximo que pueden recibir los motores, que el motor ignorará los valores superiores a T_{on}=2000 µs y podría ocurrir que el cuadricóptero no realizase todos los movimientos ordenados desde el transmisor de radio control. Por tanto, habrá que establecer estos valores como límites mínimo y máximo para adecuar la señal enviada a los motores. Por último, para mandar estas señales a los motores utilizamos la librería Servo, es decir, se realiza de la misma forma que en la inicialización de los motores.

4.3.3 Despegue seguro

A la hora de realizar el despegue vertical del quadcopter es fundamental que este situado sobre una superficie horizontal. Para asegurarnos de que no se intenta realizar el take-off en una posición inicial insegura, se ha diseñado el siguiente sistema de seguridad.

Algoritmo 4.12 Despegue seguro

Algorithm 12: Despegue seguro

Input: Las variables *Roll_imu* y *Pitch_imu* son los ángulos de Euler calculados en la unidad inercial.
Output: Ninguna

```
1 Define El método Imu_update() estima la actitud del UAV. Angle_min y Angle_max son los límites a partir
  de los cuales es seguro realizar el despegue del UAV.
2 begin
3 while Stable=FALSE do
4     Imu_update();
5     if Angle_min < Roll_imu < Angle_max  &  Angle_min < Pitch_imu < Angle_max then
6         Stable = TRUE;
      else
7         for i=1 ; i ≤ 4 ; i++ do
8             Motor[i] ← acum_M[i] µs;
9 return
```

Como podemos observar, se han establecido unos límites de inclinación del quadcopter y si éste no se encuentra entre ellos, el sistema se quedará en reposo a la espera de que se vuelva a una posición más segura. Mientras tanto se envía a los motores una referencia constante para que estos no se desarmen pero que sea insuficiente para comenzar el vuelo. Dichos límites se pueden modificar por el usuario en cualquier momento.

Por otro lado, este sistema también sirve para garantizar que aunque la unidad de medición inercial sufra un pequeño fallo y entregue unos ángulos incorrectos, el UAV no emprenda un vuelo errático en el que se podrían dañar tanto recursos humanos como materiales.

4.4 Conexionado de los elementos y diseño de la PCB

El conexionado de los elementos con la placa UDOO se ha realizado siguiendo las pautas expuestas a continuación.

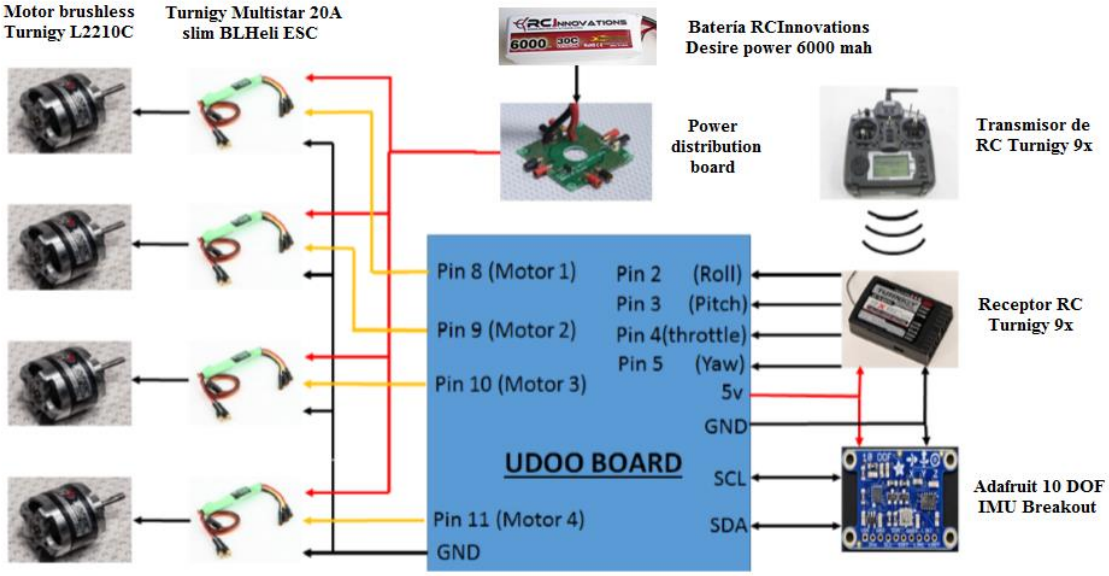

Figura 4.18 Conexionado general de los elementos

En primer lugar, la placa UDOO se alimenta con una batería LiPo externa de tres celdas y 1500 mAh, capacidad más que suficiente. A la placa UDOO se conecta el receptor de RC *Turnigy 9x*, se utilizan los pines digitales 2, 3, 4 y 5 para transmitir la información de los canales roll, pitch, throttle y yaw, respectivamente.

La unidad de medición inercial *Adafruit 10 DOF IMU Breakout* se conecta mediante los a los pines SCL y SDA puesto que se comunica por I2C. El receptor RC y la IMU se alimentan a 5V desde la placa, sin embargo, los ESC *Turnigy Multistar 20 A Slim* precisan una alimentación externa puesto que tienen un BEC integrado. Esta alimentación se distribuye a los cuatro variadores utilizando la *Power distribution board*, la cual está conectada a la batería *RCInnovations Desire power* de 6000 mAh de capacidad.

Por otro lado, las señales de control de los motores se mandan a los ESC a través de un cable triple que contiene alimentación, control y tierra. Es fundamental dejar los cables de alimentación sin conectar, ya que el ESC ya se está alimentando externamente. Los cables de control se conectan a los pines digitales de salida 8, 9, 10 y 11, que corresponden con los motores 1, 2, 3 y 4, respectivamente. Por último, los motores se conectan a los variadores electrónicos mediante tres sencillos cables, cuyo orden de conexión determina el sentido de giro del motor.

Como acabamos de explicar, en un quadcopter se integran multitud de elementos que requieren numerosos cables para realizar las interconexiones. Los cables incrementan el peso del quadcopter y pueden salirse durante el vuelo, comprometiendo la seguridad tanto del UAV como de las personas que se encuentran alrededor. Por estas razones, se ha decidido realizar una PCB que se colocará encima de la placa UDOO como si fuese un shield. El diseño de la PCB se ha realizado utilizando el programa Eagle. En la figura siguiente podemos observar las interconexiones que se han de realizar.

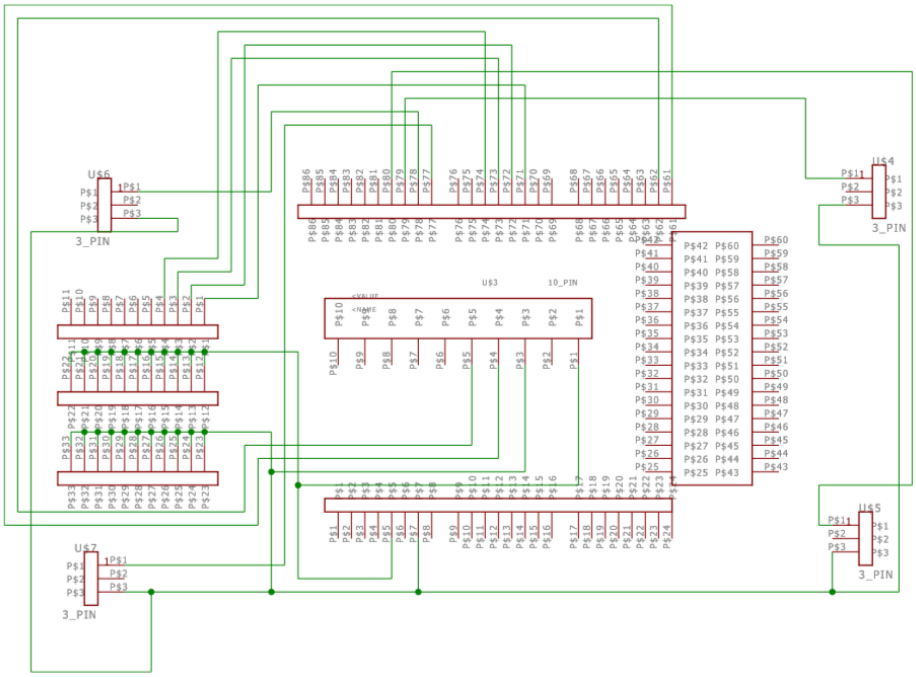

Figura 4.19 Conexiones de los elementos de la PCB

Después de realizar el proceso de emplazamiento y rutado, el aspecto de la placa PCB diseñada es el siguiente.

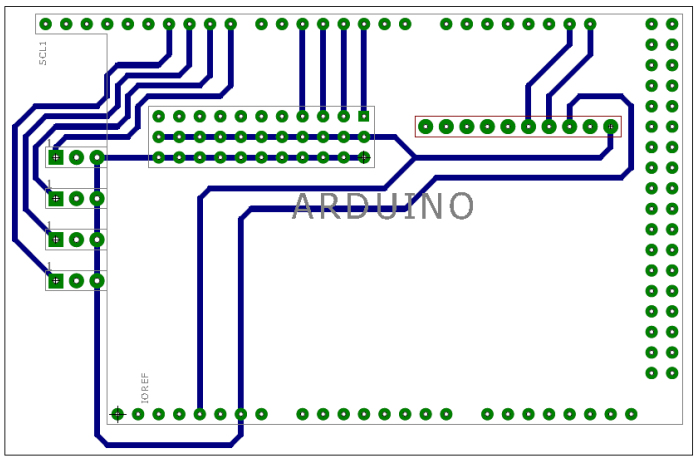

Figura 4.20 Aspecto de la PCB diseñada

Como podemos observar, en la parte exterior de la PCB se encuentran tres grandes tiras de pines colocadas en forma de U que coinciden con la morfología del Arduino Due contenido en la UDOO. En la fila de diez pines situada a la derecha de la placa se colocará la unidad de medición inercial. El conjunto de pines situado en la parte central estará dedicado al receptor de radio control Turnigy 9x. Por último, se ha colocado unos conectores triples para el conjunto de tres cables por el que se realiza el control de los motores. Éstos se han situado a la izquierda de la placa para minimizar la distancia con los pines de salida asociados a los motores, facilitando así el rutado.

A continuación se muestra el conexionado de los elementos principales del UAV con la placa UDOO utilizando la PCB diseñada.

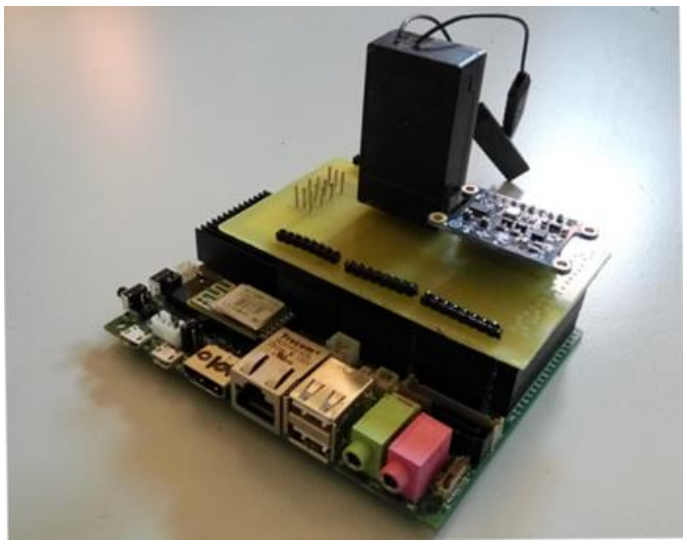

Figura 4.21 Placa UDOO con la PCB

Por último, al conectar todos los elementos del UAV definidos en "Figura 4.18", el aspecto del quadcopter diseñado es el siguiente.

Figura 4.22 Aspecto final del Quadcopter con la PCB

Capítulo 5 Diseño del Sistema de Control

Los cuadricópteros son sistemas inestables por definición. Desde un punto de vista aerodinámico, los UAV de ala fija poseen una morfología que les permite mantenerse en el aire, sin embargo la morfología de los multirotores hace que requieran un sistema de control para funcionar correctamente, de lo contrario caerían directamente contra el suelo. Los cuadricópteros tienen seis grados de libertad pero solo cuatro entradas, lo que los convierte en sistemas subactuados. Por tanto, es fundamental diseñar un sistema de control robusto que estabilice el cuadricóptero sobre una actitud deseada.

Hoy en día existen multitud de sistemas de control aplicables a los UAV como por ejemplo el control LQR o el control por linealización en la realimentación, pero hemos decidido utilizar un regulador PID puesto que es un sistema de control ampliamente utilizado en la industria gracias a su flexibilidad, efectividad y sencillez. Otros reguladores exigen un modelo muy exacto del sistema a controlar, sin embargo, el regulador PID nos permitirá controlar nuestro sistema en cualquier caso. Para más información acerca de otros sistemas de control, consultar el apartado "Estrategias de control" incluido en el estado del arte.

5.1 Fundamentos del regulador PID

5.1.1 Componentes del regulador PID

El regulador PID es un sistema de control por realimentación que se basa calcular el error cometido entre el valor medido y el valor deseado *e(t)* y ajustarlo con los tres términos que componen el regulador. Son los términos proporcional (P), integral (I) y derivativo (D). Ajustando estos tres parámetros es posible controlar prácticamente cualquier sistema.

El término Proporcional, otorga una salida proporcional al error cometido, de modo que el valor de salida *u(t)* será el error multiplicado por la ganancia K_p.

$$u(t) = K_p \cdot e(t) \tag{5.1}$$

Para ajustar este término, es preciso que la ganancia K_p sea lo suficientemente alta para controlar el sistema pero no demasiado alta, de lo contrario el sistema sobreoscilará haciéndose inestable. Cuando realizamos el control de un sensor de temperatura, es posible que el sistema se ajuste utilizando únicamente la acción proporcional, sin embargo, puesto que la dinámica de los cuadricópteros es no lineal, será necesario utilizar el regulador PID completo.

El término Integral, tiene como propósito eliminar el error en régimen estacionario, es decir, ayuda a alcanzar el valor de referencia deseado. Como se ve en la siguiente expresión, la acción integral depende de la suma de los errores pasados.

$$u(t) = K_i \cdot \int_0^t e(t)\, dt \tag{5.2}$$

De modo que el error acumulado se multiplicará por la constante integral K_i para entregar el valor de salida *u(t)*. El control con el término integral por sí solo se utiliza en sistemas muy de primer orden, por lo que no es nuestro caso. Si juntamos la acción proporcional e integral, obtendremos un regulador PI. Este tipo de controlador elimina el error estacionario pero aumenta el tiempo de establecimiento y si no está bien ajustado, puede llegar a ser inestable.

El término Derivativo, trata de minimizar el error corrigiéndolo proporcionalmente a la velocidad con la que se produce. La salida del sistema dependerá de la variación del error respecto del tiempo, por lo que trata de ajustar el error que se tiende a producir.

$$u(t) = K_d \cdot \frac{de(t)}{dt} \tag{5.3}$$

El efecto de la acción derivativa suele ser minimizar las oscilaciones producidas alrededor del punto deseado o setpoint. Sin embargo, la acción derivativa es sensible al ruido por lo que la ganancia K_d resulta difícil de ajustar.

Un regulador PID completo aúna la acción de los términos proporcional, integral y derivativo, por lo que la expresión que representa su funcionamiento es la siguiente.

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(t)\, dt + K_d \cdot \frac{de(t)}{dt} \tag{5.4}$$

De esta forma, es posible realizar el control del sistema deseado ajustando únicamente las tres ganancias, y es fácil ver el efecto que produce modificar alguna de ellas, razón por la cual el control PID se utiliza en la gran mayoría de los sistemas industriales. Las ganancias del PID se pueden ajustar manualmente, sin embargo, en sistemas complejos es preferible modelar el sistema y de esta forma realizar simulaciones que nos facilitarán considerablemente la tarea. Además, existen procedimientos diseñados para reducir el tiempo y esfuerzo requeridos.

5.1.2 Funcionamiento general del sistema de control.

El sistema de control de vuelo del UAV se basa en la estabilización de la actitud del cuadricóptero sobre unas referencias determinadas. Para controlar nuestro sistema, es necesario utilizar tres reguladores PID independientes, uno para cada ángulo de Euler. La siguiente figura ilustrará el funcionamiento del sistema de control.

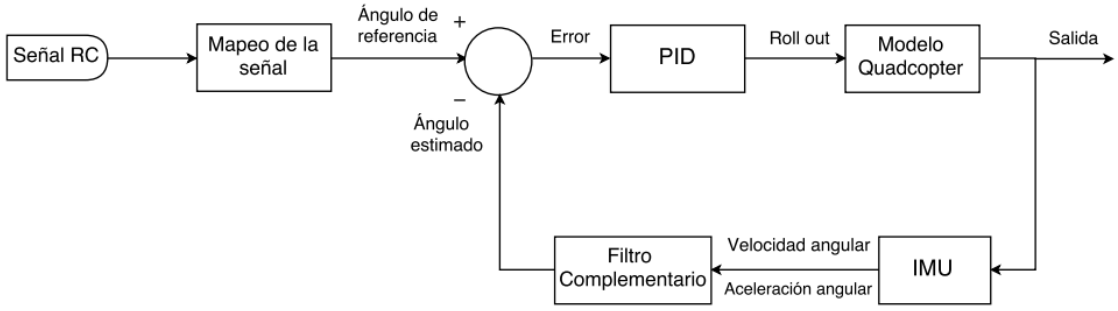

Figura 5.1 Diagrama de bloques del sistema

Las referencias deseadas se reciben a través de las señales enviadas por el transmisor de radio control. Estos comandos se han de leer y mapear correctamente para facilitar la tarea del regulador PID. Se realiza el mapeo estableciendo unos límites de inclinación del quadcopter de ±30 grados que asegurarán que la dinámica del sistema no se vuelva incontrolable.

El estado del sistema se estima gracias al uso de un filtro complementario que fusiona la información recibida por los giróscopos y acelerómetros. De este modo obtenemos una medida notablemente más fiable de los ángulos de roll, pitch y yaw, sin los problemas que presentan cada uno de los sensores por separado.

La diferencia entre el estado deseado (referencias del mando de radio control) y el estado actual estimado, da como resultado el error producido. En función del error producido, el punto de referencia deseado, los límites de trabajo y las constantes de control, el regulador PID calcula una salida que deberá ser enviada a los ESC que aumentarán o disminuirán la velocidad de los motores para estabilizar el sistema.

Ahora que tenemos una visión más generalista del sistema de control, procederemos a explicar el sistema de control a través de sencillos algoritmos. Comenzaremos por una explicación del regulador PID en lenguaje informático y los problemas que pueden producirse, para luego explicar la interacción entre los distintos algoritmos que se han explicado e implementado hasta ahora.

5.2 Regulador PID en Arduino

5.2.1 Funcionamiento general

En el presente proyecto se ha implementado un sistema de control basado en el regulador PID de la librería de Arduino. Como explicaremos a continuación, es posible diseñar un algoritmo PID muy sencillo simplemente basándonos en la expresión 5.4, sin embargo, utilizar el regulador de la librería nos aporta muchas ventajas. Algunas de ellas son que se asegura de ejecutarse regularmente, mitiga el "Derivative Kick", nos permite cambiar las constantes de control durante el vuelo, se asegura de que no haya retrasos en la señal de control, etc.

Además, el hecho de que el algoritmo PID haya sido probado y utilizado anteriormente para controlar sistemas similares, nos aporta cierta seguridad de que funcionará correctamente. Y puesto que es de código abierto, no habría problema en realizar cualquier modificación del mismo.

El funcionamiento básico de un regulador PID se puede describir utilizando el siguiente algoritmo, el cual es una simplificación del algoritmo PID utilizado.

Algoritmo 5.1 Regulador PID básico

Algorithm 13: Regulador PID básico

Input: Ángulo deseado $Setpoint$, ángulo actual $Input$
Output: Salida del regulador $Output$
1 **Define** Tiempo actual en milisegundos now, tiempo de la anterior ejecución $LastTime$, tiempo transcurrido $TimeChange$, error actual $error$, error de la anterior ejecución $LastErr$, error acumulado $errSum$, variación del error $dErr$.
2 **begin**
3 $now \leftarrow millis()$;
4 $TimeChange \leftarrow now - LastTime$;
5 $error \leftarrow Setpoint - Input$;
6 $errSum \leftarrow errSum + error \cdot TimeChange$;
7 $dErr \leftarrow (error - LastErr)/TimeChange$;
8 $Output \leftarrow kp \cdot error + ki \cdot errSum + kd \cdot dErr$;
9 $lastErr \leftarrow error$;
10 $lastTime \leftarrow now$;
11 **return**

Como podemos observar, se almacena el tiempo transcurrido en la variable *TimeChange,* se calcula el error como la diferencia entre el punto deseado y estado actual y se calculan el error acumulado y la derivada del error. Por último, los valores *error*, *errSum* y *dErr* se multiplican por las constantes de control y la suma de cada término es la salida del regulador.

Teóricamente, el algoritmo anterior funcionaría como regulador PID, pero si vamos a controlar un sistema real como el propuesto en este proyecto, será necesario utilizar un algoritmo un poco más complejo.

En el control PID es fundamental que la regulación se ejecute a intervalos regulares, de lo contrario se complicará el cálculo matemático ya que los términos derivativo e integral dependen del tiempo. De esta forma, se especificará un tiempo de muestreo y el algoritmo se encargará de evaluarse únicamente en el momento preciso.

Cuando cambiamos rápidamente la señal de referencia procedente del transmisor de radio control, el término derivativo adquiere un valor muy elevado puesto que ha sufrido una variación considerable en un tiempo muy pequeño. Éste efecto se conoce como "Derivative Kick" y provocará que la señal de salida tenga picos no deseados. Para solucionarlo, hay que reflexionar sobre la siguiente expresión.

$$\frac{de(t)}{dt} = \frac{dSetpoint}{dt} - \frac{dInput}{dt} \Rightarrow Setpoint = cte \Rightarrow \frac{de(t)}{dt} = -\frac{dInput}{dt} \qquad (5.5)$$

Por tanto si reemplazamos la derivada positiva del error por la derivada negativa de la entrada, el problema de los picos en el término derivativo desaparecerá, y como hemos visto, esto no afecta en absoluto durante el régimen estacionario de la señal setpoint.

En el control tradicional de estabilización de vuelo del UAV no es muy habitual cambiar las constantes de control durante el vuelo, pero si vamos a realizar un control más avanzado, como por ejemplo un control reactivo que mantenga el vuelo del cuadricóptero a una distancia fija de la pared, será necesario cambiarlas. Sin embargo, al cambiar dichas constantes el término integral se ve gravemente afectado ya que contiene el error acumulado y una modificación de la constante provocará que la salida fluctúe. Como necesitamos que el impacto del cambio de la Ki afecte únicamente al error que se acumulará a partir del momento del cambio, se procederá a escalar el error acumulado en un factor 1/Ki eliminando así el problema.

Otro efecto importante es el windup, éste se produce cuando tenemos un gran error durante mucho tiempo lo que hará que los términos del PID, sobretodo el integral, sigan aumentando para compensar el error sobrepasando así los límites reales los motores. Esto se soluciona estableciendo unos límites de salida al regulador PID y en caso de que éstos se sobrepasen, imponer el valor máximo para que la señal de salida no sufra retrasos.

En definitiva, el regulador PID incluido en la librería de Arduino tiene en cuenta, entre otros, los efectos mencionados anteriormente, lo que pone de manifiesto que es un algoritmo bastante completo.

5.2.2 Algoritmo completo

Ahora que conocemos el funcionamiento general del regulador PID, es el momento de configurarlo. Debemos recordar que vamos a utilizar tres reguladores independientes para controlar los ángulos de Euler. Para facilitar la explicación se han inicializado todos los reguladores con los mismos parámetros, pero en la realidad no tiene por qué ser así.

Algoritmo 5.2 Inicialización del regulador PID

Algorithm 14: PID_initialize

Input: Ninguna
Output: Ninguna

```
1 Define controller[1], controller[2],controller[3] son reguladores PID asociados a los ángulos Roll, Pitch y
  Yaw, respectivamente. PID_Min y PID_Max son los límites de salida del regulador. SampleTime es el
  tiempo de muestreo. Las constantes de control son kp, ki, kd.
2 begin
3 for i=1 ; i ≤ 3 ; i++ do
4     controller[i].SetOutputLimits(PID_Min, PID_Max);
5     controller[i].SetMode(AUTOMATIC);
6     controller[i].SetSampleTime(SampleTime);
7     controller[i].SetTunings(kp, ki, kd);
8 return
```

En primer lugar debemos especificar los límites máximo y mínimo para la señal de salida del regulador. En nuestro caso el rango de salida es de ±200. Como el rango de operación de los motores se ha establecido entre 1100 µs y 2000 µs resulta difícil sobrepasarlo. También es necesario especificar el modo automático para que el regulador funcione correctamente. El tiempo de muestreo o *SampleTime* es fundamental puesto que determina el periodo fijo en el que se evalúa el regulador. Éste debe ser lo más pequeño posible para facilitar el control, en nuestro caso es de 2.8 ms. En otras palabras, el regulador PID se ejecuta regularmente con una frecuencia de 350 Hz. Por último, hay que especificar las constantes de control. Puesto que el cuadricóptero que utilizamos es simétrico, las constantes del *Roll_controller* y *Pitch_controller* son idénticas.

Como el *SampleTime* es fijo, podemos escalar las constantes de control y no será necesario incluir la variable tiempo en el algoritmo completo, simplificando así los cálculos matemáticos. Esto se ha realizado en el algoritmo siguiente.

Algoritmo 5.3 Función SetTunings()

Algorithm 15: SetTunings()

Input: $newKp$, $newKi$, $newKd$
Output: kp, ki, kd

```
1 Define El tiempo de muestreo en unidades del sistema internacional (segundos) es SampleTimeInSec.
2 begin
3 if newKp < 0 or newKi < 0 or newKd < 0 then
4     return
5   kp ← newKp;
6   ki ← newKi · SampleTimeInSec;
7   kd ← newKd/SampleTimeInSec;
8 return
```

Finalmente, el algoritmo que detalla el funcionamiento del regulador PID completo, incluyendo los efectos anteriormente mencionados es el siguiente.

Algoritmo 5.4 Regulador PID completo

Algorithm 16: Regulador PID completo

Input: Ángulo deseado $Setpoint$, ángulo actual $Input$
Output: Salida del regulador $Output$

Define Tiempo actual en milisegundos now, tiempo de la anterior ejecución $LastTime$, tiempo transcurrido $TimeChange$, tiempo de muestreo $SampleTime$, error actual $error$, término integral acumulado $ITerm$, límite máximo de salida $outMax$, límite mínimo de salida $outMin$, ángulo de entrada de la anterior ejecución $LastInput$, variación del ángulo de entrada $dInput$.

begin
$now \leftarrow millis()$;
$TimeChange \leftarrow now - LastTime$;
if $timeChange > SampleTime$ **then**
  $error \leftarrow Setpoint - Input$;
  $ITerm \leftarrow ITerm + ki \cdot error$;
  if $ITerm > outMax$ **then**
    $ITerm \leftarrow outMax$;
  else
    if $ITerm < outMin$ **then**
      $ITerm \leftarrow outMin$;
  $dInput \leftarrow Input - LastInput$;
  $Output \leftarrow kp \cdot error + ITerm - kd \cdot dInput$;
  if $Output > outMax$ **then**
    $Output \leftarrow outMax$;
  else
    if $Output < outMin$ **then**
      $Output \leftarrow outMin$;
  $lastInput \leftarrow Input$;
  $lastTime \leftarrow now$;
return

En el caso de querer ahondar más en el algoritmo, se puede encontrar una explicación más detallada en la referencia [38].

5.3 Sistema de control completo (Flujograma general)

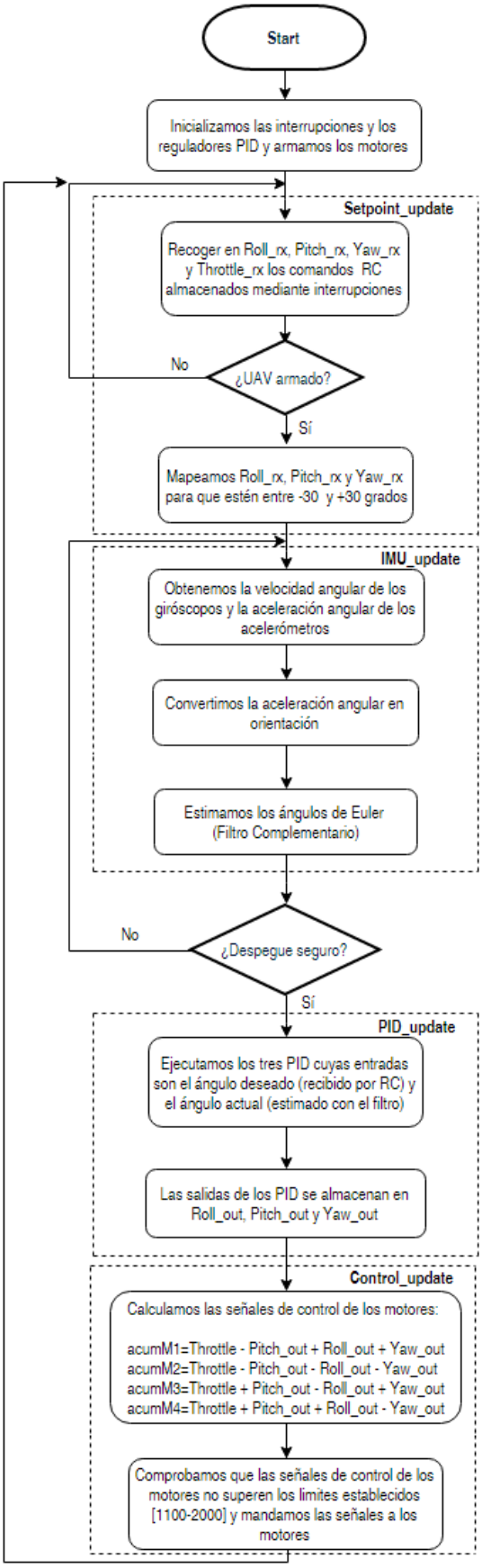

Figura 5.2 Flujograma del sistema de control

En el flujograma de la "Figura 5.2" podemos observar que el sistema de control propuesto consta de cinco procesos. En primer lugar, se realiza la inicialización de las interrupciones y reguladores PID que coinciden con las explicadas en "Algoritmo 4.2" y "Algoritmo 5.2", respectivamente. Tampoco debemos olvidar el armado de los motores descrito en "Algoritmo 4.9".

Una vez realizado esto, pasamos al proceso *Setpoint_update* que guarda en las variables *Roll_rx*, *Pitch_rx*, *Yaw_rx* y *Throttle_rx* los valores captados en las rutinas de atención a la interrupción como la descrita en el "Algoritmo 4.3" y en el siguiente flujograma.

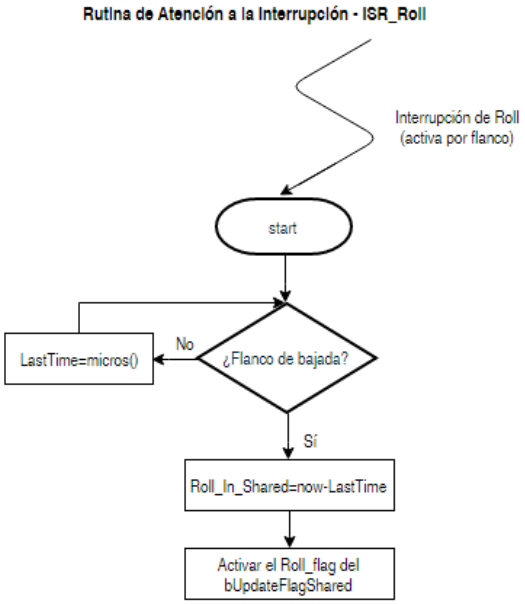

Figura 5.3 Flujograma ISR_Roll

A continuación, se comprueba si se ha armado el UAV con el proceso descrito en el "Algoritmo 4.10". De no ser así, el sistema completo se detiene y únicamente se envían señales a los motores para que éstos no se desarmen del todo. Acto seguido, se escala los comandos recibidos en *Setpoint_update* para que no se sobrepase el rango de funcionamiento establecido en ±30°.

En el proceso *IMU_update* se obtienen los datos de los giróscopos gracias al "Algoritmo 4.5" y los de los acelerómetros gracias al "Algoritmo 4.6". Con esta información se realiza una fusión sensorial gracias al filtro complementario explicado en "Algoritmo 4.7", obteniendo así la estimación de los ángulos de Euler. Ahora se evalúa si el cuadricóptero se encuentra en una posición segura para realizar el despegue (±5 grados).

Si es así, pasamos al apartado *PID_update* en el que se ejecutan el "Algoritmo 5.4" de los reguladores PID de los ángulos roll, pitch y yaw. Como consecuencia obtendremos las señales de salida para cada movimiento *Roll_out*, *Pitch_out* y *Yaw_out*.

Finalmente, en *Control_update* se calcula la señal que ha de ser enviada a cada uno de los motores mediante la suma y o resta de los componentes asociados a cada movimiento, lo cual se explica en "Algoritmo 4.11". Por último, se comprueba que las señales de los motores no superen el rango de trabajo establecido en [1100 µs - 2000 µs]. Una vez hecho esto, se vuelve al proceso *Setpoint_update* comenzando la siguiente ejecución del programa.

Capítulo 6 Modelado del cuadricóptero

Disponer del modelo de nuestro cuadricóptero es fundamental ya que nos permite elegir el sistema de control más adecuado y en base a éste se pueden realizar multitud de simulaciones que facilitarán considerablemente nuestro trabajo. Además, el modelo matemático conforma los cimientos sobre los que se asentará el control autónomo del cuadricóptero. Las ecuaciones del modelo matemático, por muy enrevesadas que puedan parecer, no son el origen de las dificultades. Éstas radican en el hecho de que el modelo no siempre representa nuestro UAV como debiera.

Por una parte, el modelo matemático utiliza aproximaciones en las que se desprecian algunos efectos aerodinámicos y por otra parte, el modelo está basado en las fuerzas de empuje de los motores, las cuales dependen de algunos parámetros como el coeficiente de empuje, que son difíciles de calcular. También utiliza el valor de los momentos de inercia que dependen de las dimensiones, peso y la localización de los elementos integrados en el cuadricóptero. Es posible calcular los momentos de inercia, si disponemos de los archivos de diseño en CAD el programa nos calculará los valores exactos, sin embargo, cada vez que incluyamos un elemento nuevo al UAV habría que modificar el diseño. Otra forma de realizarlo es a la antigua usanza, es decir, apoyándose en las masas y dimensiones de los componentes para realizar un cálculo aproximado.

Es importante recordar que el comportamiento aerodinámico del quadcopter es no lineal, por lo que en el modelado debemos realizar la linealización del mismo, esto se logra trabajando en torno a un punto de trabajo muy concreto (hover) que es el punto en el que el multirotor se sostiene en el aire y que un incremento en el empuje de los motores provocaría movimiento. Por tanto, si queremos que el modelo siga siendo representativo no debemos alejarnos demasiado de dicho punto, así pues, trabajar con ángulos de roll y pitch mayores que 30 grados podría provocar un efecto no deseado.

Ahora que conocemos las principales causas de error en el modelado, vamos a implementarlo en Matlab, lo que nos permitirá realizar multitud de simulaciones para ajustar sistema de control y conocer el comportamiento esperado del quadcopter.

En este sentido, se ha utilizado el modelo en Simulink “Quadcopter Dynamic Modeling and Simulation” desarrollado por el equipo de trabajo “Team 37” de la universidad de Drexel, quien lo ha declarado de libre acceso permitiéndonos así utilizarlo y modificarlo para adaptarlo a nuestro sistema.

La razón por la que se ha utilizado este modelo en Matlab es que no solo permite controlar la actitud (ángulos de Euler para estabilizar el cuadricóptero), sino que incluye un modelo de sistema de control para realizar el seguimiento de la posición, lo que se conoce comúnmente como "tracking position". Este nos será de gran utilidad cuando convirtamos nuestro quadcopter en un sistema que incorpore el vuelo autónomo.

Además, el modelo incorpora ciertos aspectos que representan los problemas reales que podemos encontrar al implementarlo en un sistema real que utilice una unidad de medición inercial para la estimación de la actitud y un microprocesador Arduino Due, el cual coincide con el incorporado en la placa UDOO. Algunos de estos problemas son la frecuencia de muestreo de la IMU, la cual está directamente relacionada con la capacidad de controlar el sistema, la frecuencia de trabajo de los variadores y el hecho de que no podemos crear una señal continua de salida hacia los motores, sino que estará compuesta por pequeños escalones.

Por otra parte, es posible realizar simulaciones sobre efectos aerodinámicos externos gracias al bloque "external disturbances" para comprobar cómo de sensible es nuestro sistema a alteraciones del entorno.

A continuación, vamos a explicar a grandes rasgos los distintos bloques que conforman el modelo en Simulink. En la imagen inferior se presenta el sistema completo a alto nivel.

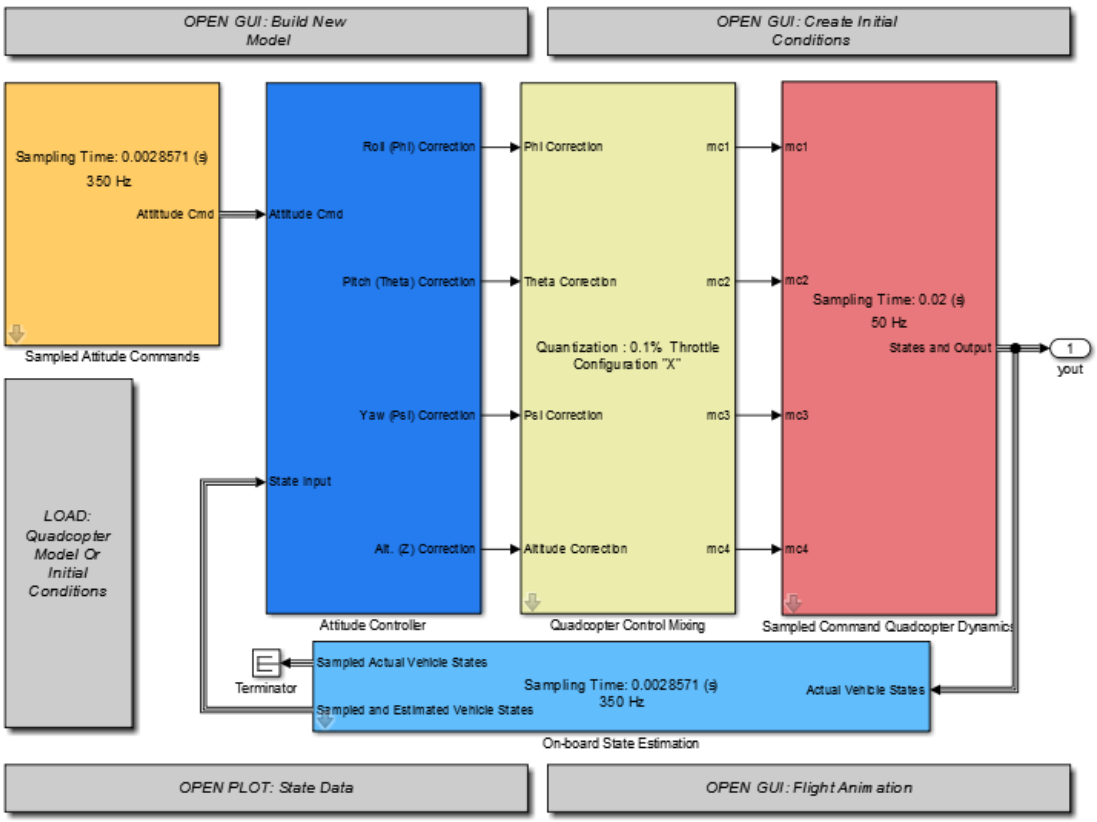

Figura 6.1 Diagrama de bloques del modelo en Simulink a alto nivel [36]

Como podemos observar, el modelo está compuesto por cinco bloques. El bloque "Sampled Attitude Commands" se encarga de mandar los comandos de actitud es decir, los ángulos de roll, pitch y yaw deseados. Este bloque simula los comandos enviados por un transmisor de radio control, por lo que es donde incluiremos los escalones de referencia para comprobar si el sistema es capaz de estabilizarse en el ángulo deseado.

El bloque "Attitude Controller" recibe los comandos de actitud y el estado (posición y orientación del UAV), evalúa el error cometido en cada uno de los ángulos y lo ajusta gracias a los reguladores PID que incluye. Es el bloque de control en sí, en el que tendremos que ajustar las ganancias del PID para estabilizar el sistema y obtener la respuesta deseada. Puesto que vamos a controlar tres ángulos, tendremos que ajustar tres reguladores, aunque el del roll y pitch serán iguales puesto que el quadcopter es simétrico.

En "Quadcopter Control Mixing" se tiene en cuenta la configuración del cuadricóptero (Quad X o Quad +) y se envían las señales de control a cada uno de los motores. Esto nos permitirá hacer simulaciones con ambas configuraciones. Luego en "Sampled Command Quadcopter dinamics" las señales de control se convierten en velocidades de rotación de los motores que entran en el modelo matemático, el cual calcula el estado (ángulos de Euler) y las salidas del sistema (velocidad angular, velocidad lineal y posición en los ejes x, y, z). En el caso de querer añadir perturbaciones aerodinámicas externas, éstas se incluirán en este mismo bloque.

Por último, en el bloque "Onboard State Stimation" se simula el proceso de estimación de la actitud utilizando una IMU y los algoritmos de filtrado para obtener el estado estimado del UAV. Como explicaremos más adelante, la frecuencia de muestreo de la IMU adquiere una gran importancia dado que condiciona el control del sistema.

Cabe destacar que es posible crear unas condiciones iniciales de simulación que se adapten a las de nuestro sistema. Esto se puede realizar muy fácilmente gracias a la interfaz gráfica del modelo en Simulink, que nos permite especificar la actitud, las velocidades lineales y angulares, la posición y la velocidad de rotación de los motores. En nuestro caso se han utilizado unas condiciones iniciales correspondientes al estado "Hover", que es el punto de trabajo sobre el que se linealiza nuestro modelo.

6.1 Caracterización del modelo del cuadricóptero

Como hemos comentado anteriormente, para que el modelo sea representativo es necesario definir correctamente tanto los momentos de inercia del cuadricóptero, como los parámetros del motor. Para facilitar esta tarea, el modelo en Simulink dispone de una interfaz gráfica en la que se deben especificar las dimensiones y peso de los componentes, y el programa calculará automáticamente los momentos de inercia del sistema.

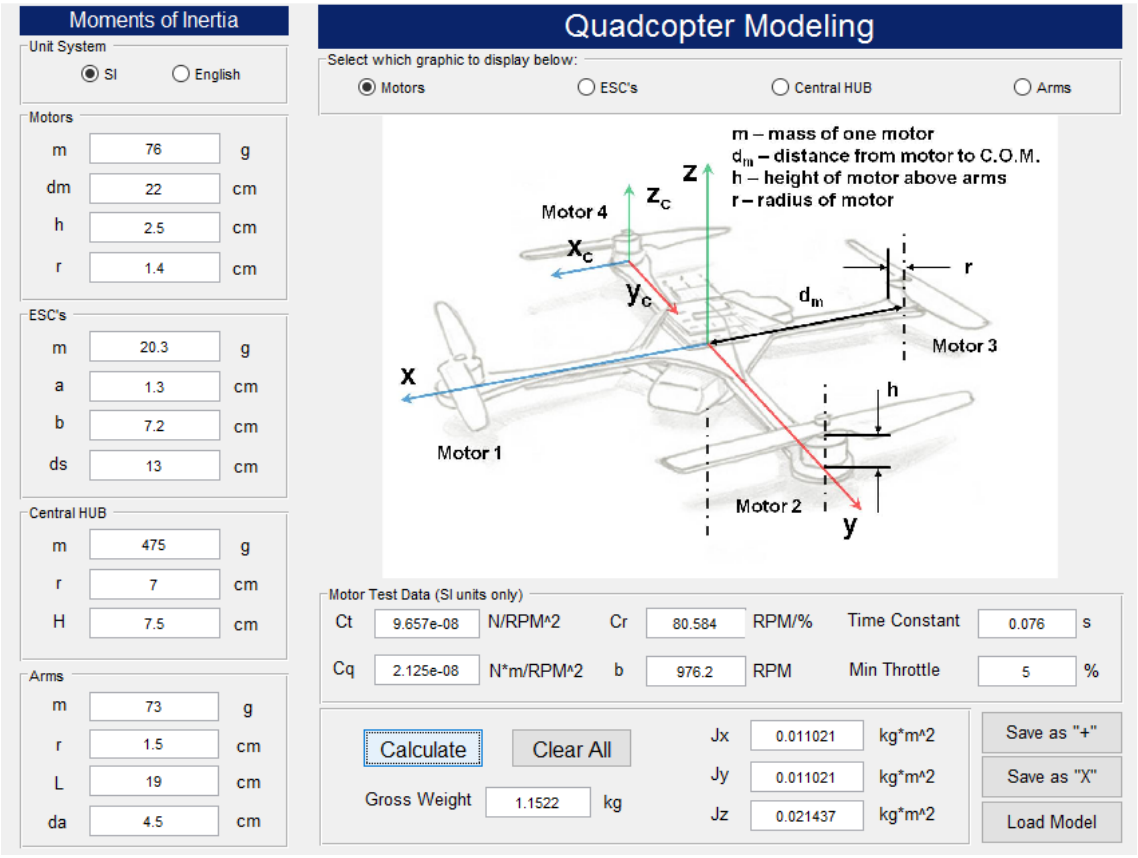

Figura 6.2 Caracterización del modelo físico del cuadricóptero [36]

Después de introducir los datos requeridos de dimensiones y peso, los momentos de inercia resultantes que definen nuestro UAV son:

- $I_x = 1{,}1021 \cdot 10^{-2}\ \mathrm{Kg} \cdot \mathrm{m}^2$
- $I_y = 1{,}1021 \cdot 10^{-2}\ \mathrm{Kg} \cdot \mathrm{m}^2$
- $I_z = 2{,}1437 \cdot 10^{-2}\ \mathrm{Kg} \cdot \mathrm{m}^2$

En cuanto a los motores, es necesario introducir los parámetros C_t , C_q , C_r , b, t. Para calcularlos es necesario medir la velocidad angular, empuje, par y constante de tiempo Éstas medidas se han obtenido de unas pruebas de eficiencia del motor en cuestión, concretamente de la referencia [37] contenida en la bibliografía.

En la siguiente tabla se representan las medidas de velocidad angular y empuje obtenidas al realizar pruebas del motor Turnigy L2210C utilizando distintas hélices y con tensiones de alimentación distintas.

Turnigy L2210C-1200Kv

Prop	Volts	Amps	Watts	rpm	Thrust	Efficiency
-	7.0	0.79	5.5	8586	-	-
GWS EP8040	7.0	5.1	36	7523	291g	74%
GWS HD8040x3	7.0	5.7	40	7397	306g	74%
GWS EP9050	7.0	8.4	59	6762	449g	71%
APC 9x6E	7.0	10.5	73	6278	470g	68%
GWS EP1060	7.0	11.1	78	6144	530g	67%
GWS HD9075	7.0	11.7	81	6019	427g	65%
-	10.5	0.95	10	12874	-	-
APC 7x4E	10.5	8.0	85	10981	460g	74%
APC 7x5E	10.5	9.4	98	10598	489g	73%
GWS EP8040	10.5	9.5	100	10575	596g	73%
GWS HD8040x3	10.5	10.7	113	10255	615g	72%
GWS EP9050	10.5	14.7	154	9182	830g	66%

Figura 6.3 Medidas características del motor Turnigy L2210C [37]

Las hélices seleccionadas en el apartado 3.2 son de 9 pulgadas con un paso de 4,7 por lo que los datos que utilizaremos serán los correspondientes a la última prueba. Con estos datos podemos calcular los parámetros más relevantes, los cuales son el coeficiente C_t relativo al empuje del motor respecto a la velocidad angular al cuadrado, y el coeficiente C_q que representa el par entregado por el motor con respecto a la velocidad angular al cuadrado.

$$C_t = \frac{Empuje}{\omega^2} = \frac{0{,}83 \cdot 9{,}81}{9182^2} = 9{,}657 \cdot 10^{-8}\ \frac{N}{rpm^2} \tag{6.1}$$

$$C_q = \frac{Par}{\omega^2} = \frac{0{,}83 \cdot 9{,}81 \cdot 0{,}22}{9182^2} = 2{,}125 \cdot 10^{-8}\ \frac{N \cdot m}{rpm^2} \tag{6.2}$$

El resto de parámetros se dejan a su valor por defecto puesto que no tenemos suficientes datos para calcularlos y porque no afectan en el modelo matemático para obtener las fuerzas de empuje ni los momentos de inercia.

Por último se guarda la configuración y se importa la caracterización del modelo en el bloque situado en la esquina inferior izquierda de la "Figura 6.1"

6.2 Fundamentos matemáticos del modelado

6.2.1 Ejes de referencia y matrices principales

El modelo matemático describe el comportamiento físico de la aeronave a través de una serie de ecuaciones. Dicho modelo está basado en las dinámicas correspondientes a la orientación del cuadricóptero, por lo que se utilizan las ecuaciones de Newton-Euler en las que se asume que el comportamiento del cuadricóptero es equivalente al de un sólido rígido. A la hora de analizar el modelo, es fundamental conocer la configuración del cuadricóptero, que determinará la disposición de los ejes del sistema de referencia. En nuestro caso, se ha seleccionado una configuración en X como se muestra a continuación.

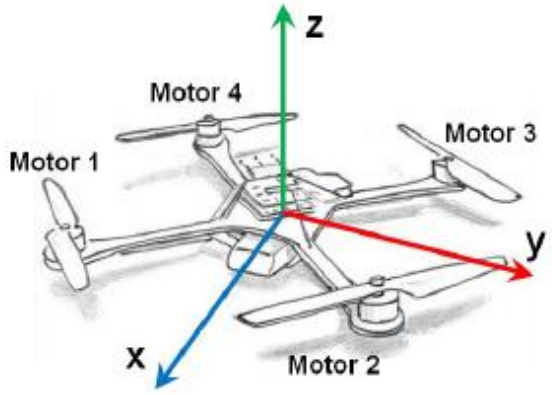

Figura 6.4 Sistema de referencia del modelado de un Quad X [36]

Como se puede ver, en este tipo de configuración los ejes de referencia no coinciden con los brazos del quadcopter, por lo que el modelo matemático de ningún modo puede ser igual que el de uno con configuración en +. Para modelar nuestro sistema, es necesario otro sistema de referencia fijo en el espacio.

En el modelo matemático se utilizarán multitud de símbolos para representar los ángulos, las velocidades lineales y angulares. Éstos se han resumido en la siguiente figura.

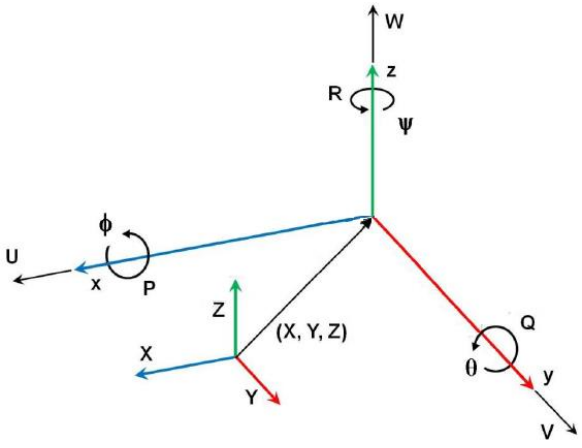

Figura 6.5 Principales símbolos asociados al sistema de referencia [36]

Nomenclatura:

(Φ, θ, ψ) : Ángulos de Roll, Pitch, Yaw

(P, Q, R) : Velocidad angular respecto de los ejes x, y, z

(U, V, W) : Velocidad lineal respecto de los ejes x,y,z

En el caso de que tengamos que referir alguna de las magnitudes del sistema de referencia móvil al sistema de referencia fijo utilizaremos la siguiente matriz de transformación.

$$R = \begin{bmatrix} \mathrm{C}_\psi C_\theta & \mathrm{C}_\psi S_\theta S_\phi - \mathrm{S}_\psi C_\phi & \mathrm{C}_\psi S_\theta C_\phi + \mathrm{S}_\psi S_\phi \\ \mathrm{S}_\psi C_\theta & \mathrm{S}_\psi S_\theta S_\phi + \mathrm{C}_\psi C_\phi & \mathrm{S}_\psi S_\theta C_\phi - \mathrm{C}_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix} \tag{6.3}$$

En el caso contrario, podemos referir magnitudes del sistema de referencia fijo al móvil realizando la inversa de la matriz anterior que, al ser ortogonal, coincide con la matriz traspuesta.

$$R^{-1} = \begin{bmatrix} \mathrm{C}_\psi C_\theta & S_\phi C_\theta & -S_\theta \\ \mathrm{C}_\psi S_\theta S_\phi - \mathrm{S}_\psi C_\phi & \mathrm{S}_\psi S_\theta S_\phi + \mathrm{C}_\psi C_\phi & C_\theta S_\phi \\ \mathrm{C}_\psi S_\theta C_\phi + \mathrm{S}_\psi S_\phi & \mathrm{S}_\psi S_\theta C_\phi - \mathrm{C}_\psi S_\phi & C_\theta C_\phi \end{bmatrix} \tag{6.4}$$

Puesto que la estructura del cuadricóptero es simétrica, la matriz de los momentos de inercia será una matriz diagonal. Idealmente los términos $\mathrm{I_{xx}}$ e $\mathrm{I_{yy}}$ deberían ser iguales, sin embargo en un sistema real esto no siempre es verdad por lo que la matriz será:

$$\mathrm{I} = \begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix} \tag{6.5}$$

A continuación se exponen las expresiones que representan los empujes y los pares del sistema. En ellas se utilizará el coeficiente de empuje $\mathrm{C_t}$ y el coeficiente de par $\mathrm{C_q}$ calculados aplicando las expresiones (6.1) y (6.2).

$$\sum Thrust = C_t \cdot (w_1^2 + w_2^2 + w_3^2 + w_4^2) \tag{6.6}$$

$$\tau_\phi = \underbrace{d_x C_t \cdot (w_2^2 + w_3^2 - w_1^2 - w_4^2)}_{Par\ provocado\ por\ el\ empuje} + \underbrace{I_m Q \frac{\pi}{30} \cdot (w_1 + w_3 - w_2 - w_4)}_{Par\ giroscópico} \tag{6.7}$$

$$\tau_\theta = \underbrace{d_x C_t \cdot (w_3^2 + w_4^2 - w_1^2 - w_2^2)}_{Par\ provocado\ por\ el\ empuje} + \underbrace{I_m P \frac{\pi}{30} \cdot (w_2 + w_4 - w_1 - w_3)}_{Par\ giroscópico} \tag{6.8}$$

$$\tau_\psi = C_q \cdot (w_2^2 + w_4^2 - w_1^2 - w_3^2) \tag{6.9}$$

Donde $\mathrm{I_m}$ es la inercia de la rotación de los motores que provoca la fuerza giroscópica y $\mathrm{d_x}$ es la distancia del motor a los ejes de referencia, que puede expresarse como:

$$d_x = d_{M\to CG} \cdot sen(45) \tag{6.10}$$

Siendo $d_{M\to CR}$ la distancia del motor al centro del sistema de referencia.

Agrupando las expresiones (6.6), (6.7) y (6.8), podemos obtener la matriz de los momentos en el sistema de referencia móvil.

$$M_{A,T} = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} d_x C_t \cdot (w_2^2 + w_3^2 - w_1^2 - w_4^2) + I_m Q \frac{\pi}{30} \cdot (w_1 + w_3 - w_2 - w_4) \\ d_x C_t \cdot (w_3^2 + w_4^2 - w_1^2 - w_2^2) + I_m P \frac{\pi}{30} \cdot (w_2 + w_4 - w_1 - w_3) \\ C_q \cdot (w_2^2 + w_4^2 - w_1^2 - w_3^2) \end{bmatrix} \tag{6.11}$$

Y la matriz de fuerzas de empuje en el sistema de referencia móvil.

$$F_{A,T} = \begin{bmatrix} 0 \\ 0 \\ C_t \cdot (w_1^2 + w_2^2 + w_3^2 + w_4^2) \end{bmatrix} \tag{6.12}$$

6.2.2 Ecuaciones de estado

Teniendo en cuenta la dinámica del sólido rígido, podemos expresar que:

$$\sum M = I \cdot \alpha \Rightarrow M_{A,T} - (\Omega_{b|i} \cdot I \cdot \omega_{b|i}) = I \cdot \dot{\omega}_{b|i} \tag{6.13}$$

Siendo $\dot{\omega}_{b|i} = (\dot{P} \quad \dot{Q} \quad \dot{R})^T$ la aceleración angular del sistema móvil respecto del fijo. De donde podemos despejar la aceleración angular del sistema.

$$\dot{\omega}_{b|i} = I^{-1} \cdot \left[M_{A,T} - \Omega_{b|i} \cdot I \cdot \omega_{b|i}\right] \Rightarrow \dot{\omega}_{b|i} = \begin{bmatrix} \dot{P} \\ \dot{Q} \\ \dot{R} \end{bmatrix} = \begin{bmatrix} \tau_\phi / I_{xx} \\ \tau_\theta / I_{yy} \\ \tau_\psi / I_{zz} \end{bmatrix} - \begin{bmatrix} \frac{(I_{zz} - I_{yy})}{I_{xx}} \cdot QR \\ \frac{(I_{xx} - I_{zz})}{I_{yy}} \cdot PR \\ \frac{(I_{yy} - I_{xx})}{I_{zz}} \cdot PQ \end{bmatrix} \tag{6.14}$$

Utilizando la matriz de transformación de rotación R^{-1} de la expresión (6.3), la velocidad angular de la estructura $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ se puede transformar en velocidades de rotación $\omega_{b|i} = (P \quad Q \quad R)^T$ de la siguiente forma.

$$\omega_{b|i} = \begin{bmatrix} P \\ Q \\ R \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_\phi^{-1} \left(\begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_\theta^{-1} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \right) \Rightarrow \begin{bmatrix} P \\ Q \\ R \end{bmatrix} = \begin{bmatrix} \dot{\phi} - \dot{\psi} \cdot S_\theta \\ \dot{\psi} \cdot C_\theta \cdot S_\phi + \dot{\theta} \cdot C_\phi \\ \dot{\psi} \cdot C_\theta \cdot C_\phi - \dot{\theta} \cdot S_\phi \end{bmatrix} \tag{6.15}$$

Donde R_ϕ es la matriz de rotación cuando $\theta = 0$ y $\psi = 0$, y R_θ es la matriz de rotación cuando $\phi = 0$ y $\psi = 0$.

Despejando (6.15) obtenemos la siguiente expresión.

$$\dot{\Phi} = H(\Phi) \cdot \omega_{b|i} \Rightarrow \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & t_\theta \cdot s_\phi & t_\theta \cdot c_\phi \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi / c_\theta & c_\phi / c_\theta \end{bmatrix} \cdot \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \tag{6.16}$$

Esta expresión es conocida como ecuación cinemática de Euler. Cabe destacar que al evaluar la expresión para los ángulos de $\theta = 90°$ y $\theta = -90°$ se produce una singularidad conocida como "Gimbal-lock". Esto se puede evitar utilizando cuaternios en lugar de ángulos de Euler.

En cuanto a la expresión que describe la aceleración lineal del sistema:

$$\dot{V}_{CM|i} = \begin{bmatrix} \dot{U} \\ \dot{V} \\ \dot{W} \end{bmatrix} = \frac{1}{m} \cdot F_{A,T} + g_{b|i} - \Omega_{b|i} \cdot \omega_{CM|i} \tag{6.17}$$

Siendo $g_{b|i}$ la aceleración de la gravedad respecto del sistema de referencia móvil, para ello es necesario utilizar la matriz de transformación R^{-1}.

Por último, la expresión que determina la velocidad lineal del sistema respecto del sistema de referencia fijo.

$$\dot{P}_{CM|i} = \begin{bmatrix} X \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = R \cdot \dot{V}_{CM|i} \tag{6.18}$$

6.3 Sistema de control

Ahora que hemos caracterizado el modelo del cuadricóptero y conocemos los fundamentos matemáticos, es el momento ajustar el sistema de control y realizar una serie de simulaciones que nos permitirán analizar la respuesta del sistema. Como hemos explicado anteriormente, el sistema de control se realiza en el bloque "Attitude Controller", que está compuesto por tres reguladores PID que ajustan los ángulos de Euler hasta alcanzar la referencia deseada. Ésta se especificará en el bloque "Sampled Attitude Commands".

Si accedemos al interior del bloque "Attitude Controller", podemos observar que hay un regulador para cada ángulo de Euler y otro para regular la altitud. De momento nos centraremos en los tres primeros puesto que son a través de los cuales se realiza la estabilización del sistema.

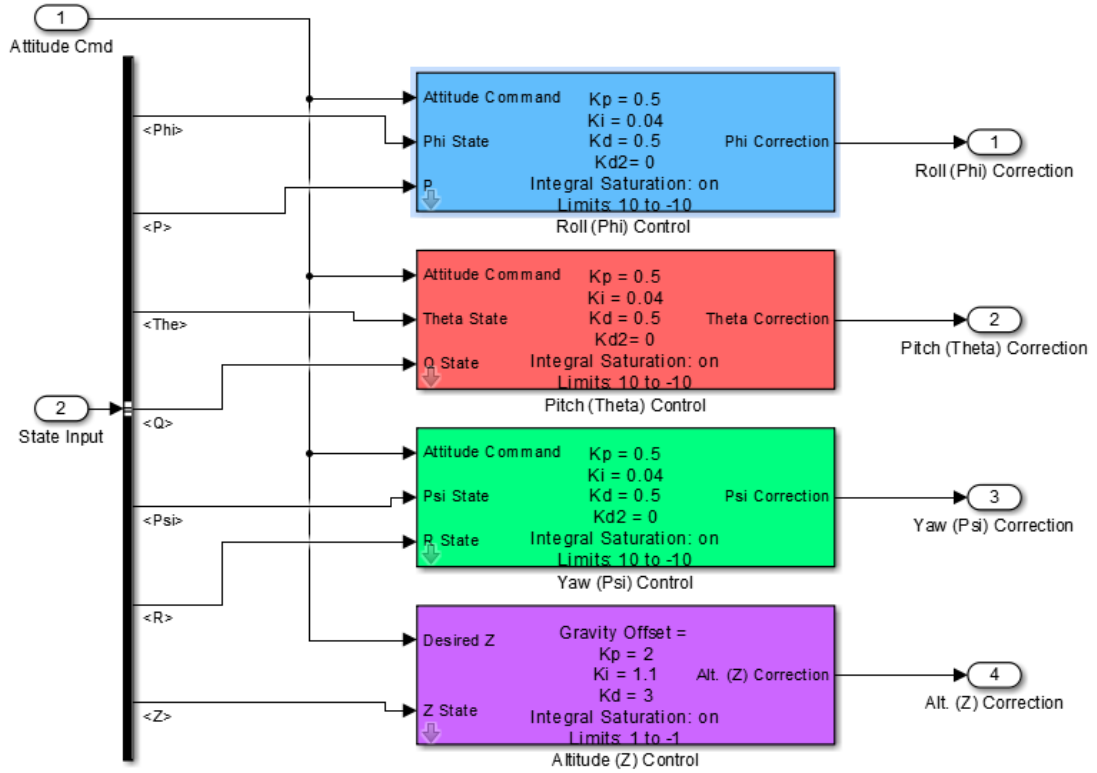

Figura 6.6 Bloque "Attitude Controller" [36]

De la imagen superior es posible obtener multitud de información. Para empezar, podemos observar que para la regulación PID se utilizan como entradas el ángulo de referencia (Attitude Cmd), los ángulos de estado y las velocidades angulares P, Q, R. A continuación, es posible modificar las ganancias K_p , K_i , K_d así como modificar los límites de saturación del término integral.

Si queremos ahondar más en el regulador PID implementado, podemos entrar en cualquiera de los bloques de la "Figura 6.6" donde nos aparecerá el siguiente diagrama.

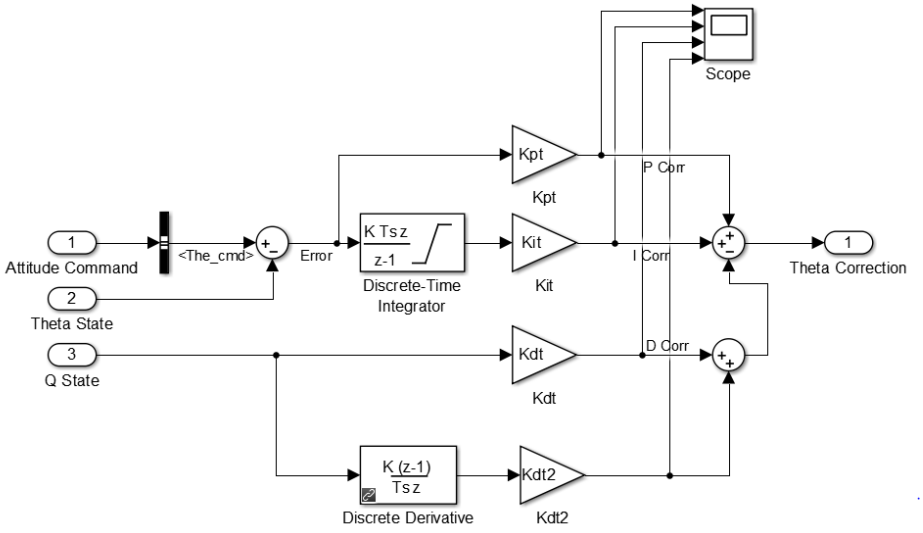

Figura 6.7 Diagrama de bloques del regulador PID implementado en Matlab [36]

El regulador implementado es ligeramente diferente al PID estándar, puesto que contiene cuatro términos. Los términos proporcional e integral se calculan evaluando el error cometido entre el ángulo de referencia y el estado actual. Por otro lado, el término derivativo K_{dt} es proporcional a la velocidad de angular, que es equivalente a utilizar la derivada del ángulo de error cometido. Por último, el término Kdt2 es proporcional a la derivada de la velocidad angular, es decir, la aceleración angular.

Teniendo en cuenta todo esto, se han ajustado los términos K_p , K_i , K_d y se ha mantenido el término K_{dt2} a un valor de cero. Tras realizar varias simulaciones, se ha ido iterando hasta obtener una respuesta adecuada. Los valores utilizados son los que se muestran en "Figura 6.6", es decir $K_p = 0{,}5$, $K_i = 0{,}04$, $K_d = 0{,}5$.

Utilizando estos valores, se ha ido al interior del bloque "Sampled Attitude Commands" y se han introducido unas referencias de 0°, 20° y 30° para los ángulos de roll, pitch y yaw, respectivamente.

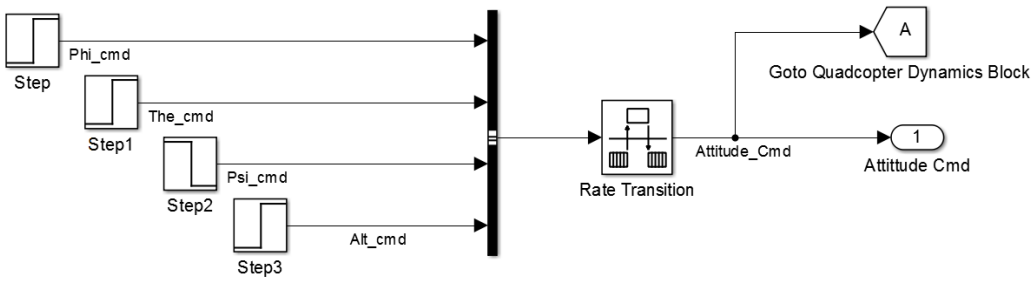

Figura 6.8 Bloque "Sampled Attitude Commands" [36]

Realizando las simulaciones se han obtenido los siguientes resultados.

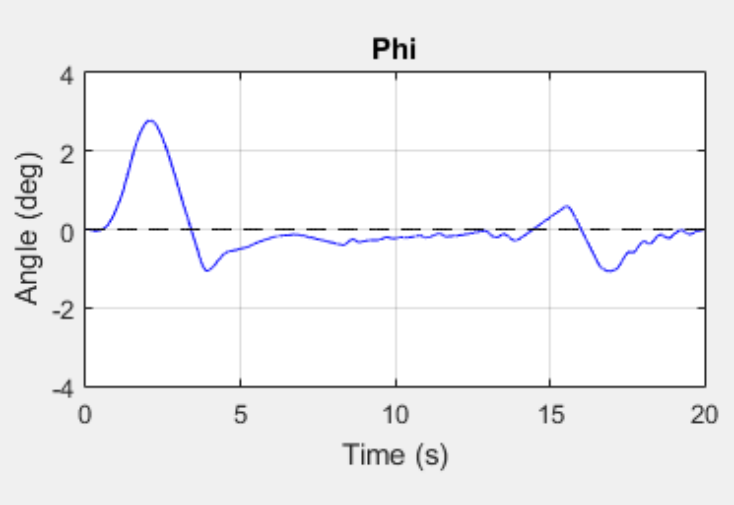

Figura 6.9 Ángulo Roll ante una referencia de 0 grados

Como podemos observar, el ángulo Phi o ángulo de roll sigue perfectamente la referencia impuesta de cero grados. En un principio, se produce una sobreoscilación de aproximadamente 3 grados que concuerda con el arranque de los motores. A partir de ese momento, el ángulo roll permanece prácticamente constante durante más de 15 segundos, con una desviación máxima de 1 grado.

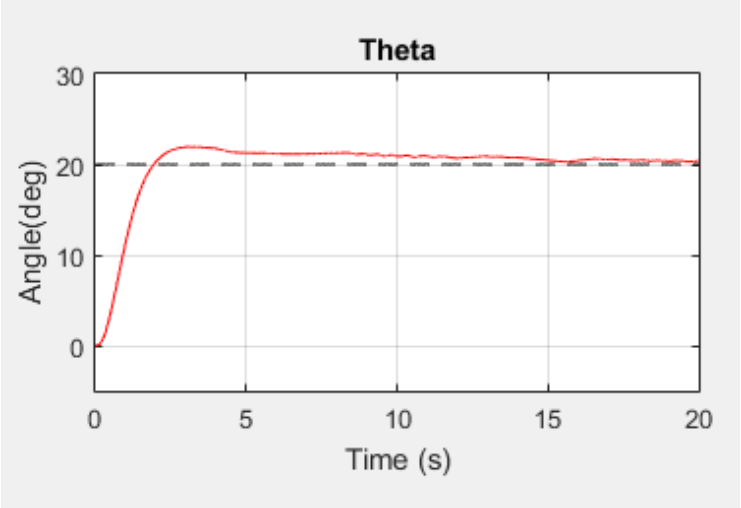

Figura 6.10 Ángulo Pitch ante un escalón de 20 grados

En este caso, se ha introducido un escalón de 20 grados en el ángulo pitch, obteniendo así una respuesta ligeramente subamortiguada, con una sobreoscilación de 1,5 grados, un tiempo de pico de aproximadamente 2,75 segundos y un tiempo de establecimiento de 3,5 segundos. Cabe destacar que se produce un ligero desfase con respecto a la referencia que con el paso del tiempo se va disminuyendo hasta alcanzar el valor exacto.

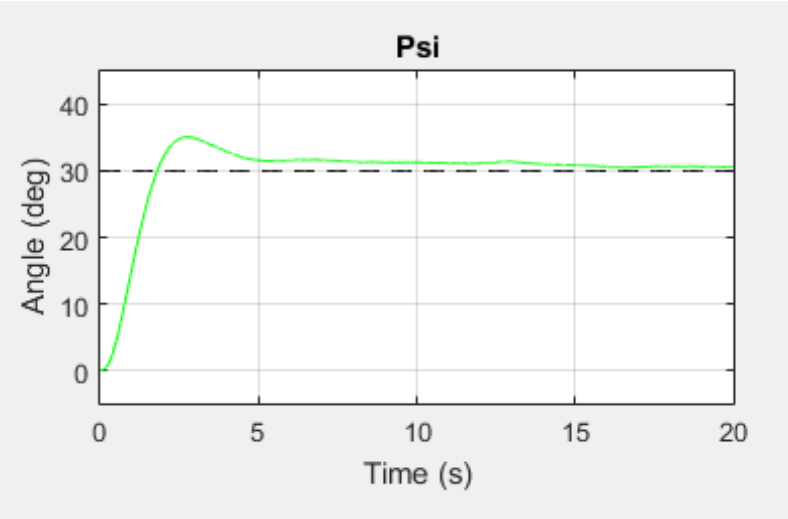

Figura 6.11 Ángulo Yaw ante un escalón de 30 grados

Al introducir un escalón de 30 grados en el ángulo yaw, se produce una respuesta subamortiguada con una sobreoscilación de 5 grados, un tiempo de pico de 2,75 segundos y un tiempo de establecimiento de alrededor de 3,75 segundos. Es decir, se ha obtenido una respuesta bastante buena, que se estabiliza prácticamente en el ángulo deseado.

También se han realizado simulaciones utilizando escalones de ángulos negativos, obteniendo los siguientes resultados.

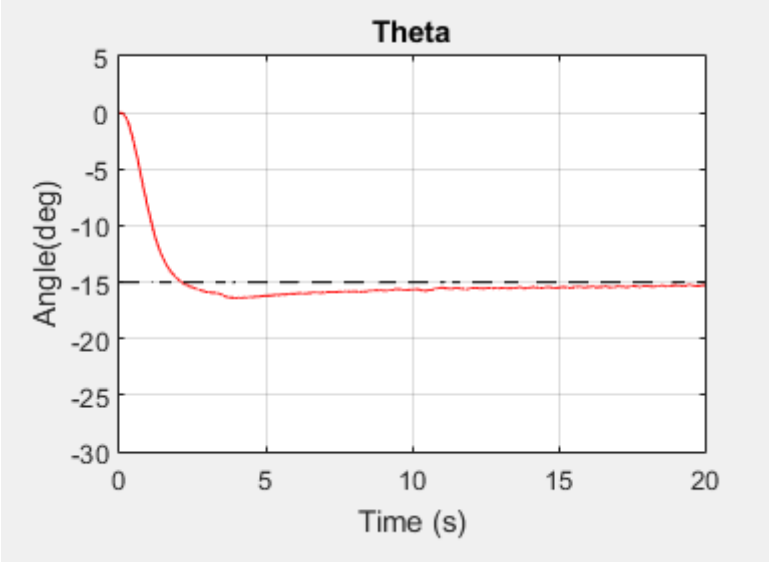

Figura 6.12 Ángulo Pitch ante un escalón de -15 grados

Ante un escalón de -15 grados de ángulo pitch, obtenemos una respuesta estable, ligeramente subamortiguada con una sobreoscilación de -1.75 grados, un tiempo de pico de 4 segundos y un tiempo de estabilización de 4,5 segundos.

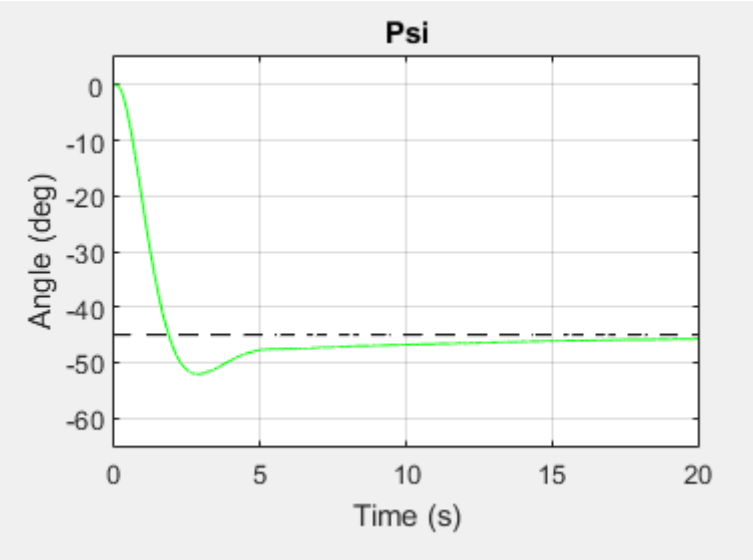

Figura 6.13 Ángulo Yaw ante un escalón de -45 grados

Por último, cuando introducimos un escalón de -45 grados para el ángulo yaw, obtenemos una respuesta subamortiguada pero que se estabiliza sobre un ángulo unos 3 grados inferior al deseado. Poco a poco, debido al efecto de la acción integral, el ángulo se va ajustando hasta obtener el valor deseado.

Basándonos en las simulaciones realizadas, podemos afirmar que todas ellas tienen un comportamiento subamortiguado y que al utilizar ángulos de referencia menores, la respuesta obtenida tiende a ser más amortiguada. Esto concuerda con el hecho de que el modelo está linealizado en torno a un punto de trabajo fijo (hover), por lo que al alejarnos de éste, obtenemos respuestas menos precisas.

Capítulo 7 Ensayos realizados

Una vez diseñado el sistema de control completo del UAV y tras realizar las simulaciones pertinentes en Matlab, es hora de realizar ensayos de vuelo en un entorno de trabajo controlado. Para ello hemos utilizado el banco de pruebas de la "Figura 7.1". En él realizaremos pruebas de estabilización del cuadricóptero limitando los grados de libertad del mismo, con el fin de ajustar cada uno de los reguladores PID por separado. Las ganancias del PID se calcularán siguiendo el método de ajuste que explicaremos más adelante. Éstas serán del orden de las constantes de control calculadas en el apartado 6.3. Por último se mostrarán los resultados obtenidos y se debatirá sobre la idoneidad de los mismos.

7.1 Banco de pruebas

En la figura siguiente se muestra el banco de pruebas utilizado. La estructura tiene que ser lo suficientemente grande para que el cuadricóptero pueda moverse con libertad, por lo que las dimensiones la misma son de 110x110x110 cm. El UAV se sujeta a la estructura mediante unas cuerdas que deben de estar centradas sobre el eje x de modo que coincida con el centro de gravedad, de no ser así el UAV se desestabilizaría.

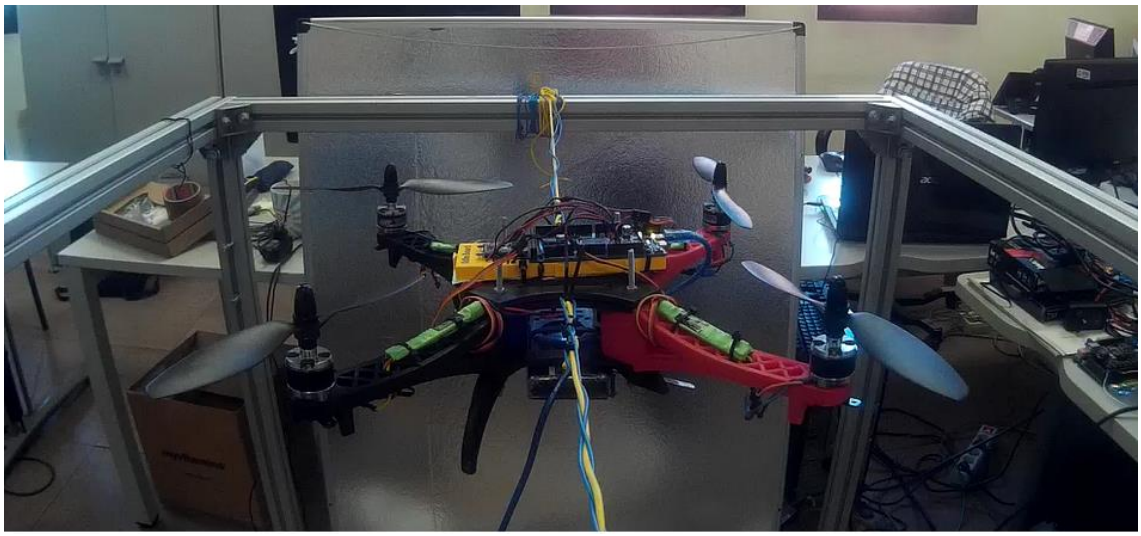

Figura 7.1 Banco de pruebas del UAV

Cuando el quadcopter está funcionando, las cuerdas que lo sujetan están sometidas a una tensión y rozamiento que podría provocar que éstas se partiesen. Para evitarlo hemos colocado unas cuerdas adicionales que no están sometidas a ninguna tensión y que actuarán de medida de seguridad.

Por último, cabe destacar que el sistema de sujeción debe estar lo suficientemente tenso para que el quadcopter realice únicamente el movimiento de rotación sobre el eje x, es decir el movimiento pitch, pero dejando el juego suficiente para que vuele con normalidad.

7.2 Procedimiento de ajuste del PID

En primer lugar, se han de limitar los grados de libertad del UAV de modo que podamos ajustar por separado los reguladores PID asociados a los movimientos de roll, pitch y yaw. Además es necesario llevar al cuadricóptero al estado de "hover", es decir, el punto en el que la fuerza de empuje vertical de los motores es igual al peso. Esto se ha conseguido enviando un comando de throttle cercano al 50%. Debemos tener en mente el orden de magnitud de las constantes de control calculadas en Matlab para hacernos una idea del valor que pueden obtener en el sistema real.

Ahora es preciso seguir el siguiente método de ajuste del regulador PID:

1. Ponemos las ganancias K_p, K_i y K_d a cero.
2. Incrementamos lentamente K_p hasta que alcanzamos la ganancia límite k_u cuya respuesta se vuelve totalmente inestable, con unas oscilaciones que cada vez son mayores. Ahora volvemos al último valor estable de K_p.
3. Incrementamos K_d hasta que las oscilaciones provocadas por el término proporcional se minimizan. Si la ganancia K_d es demasiado alta, aparecerán oscilaciones a alta frecuencia y tendremos que disminuir K_d hasta que éstas desaparezcan.
4. Lo más probable es que con el regulador PD, no alcancemos el setpoint deseado, por lo que incrementaremos la ganancia K_i hasta alcanzarlo con las mínimas oscilaciones posibles.

Siguiendo éste proceso, es posible obtener unas constantes de control adecuadas, sin embargo, no serán las óptimas. El hecho de variar el valor de una de las ganancias afecta al resto de ellas, por lo que es recomendable realizar los pasos 3 y 4 de nuevo, y comparar la respuesta del sistema.

Al realizar el ajuste manual de las constantes de control, no siempre resulta sencillo ver el efecto que tiene variar una ganancia en la estabilización del UAV, por lo que es fundamental guardar los valores del ángulo estimados en la IMU y el tiempo. De esta forma, podemos mostrar los valores en Matlab o cualquier herramienta que nos permita realizar gráficas y analizar la respuesta del sistema. Facilitando así la comparación entre distintos valores del regulador PID en cuestión.

7.3 Resultados obtenidos

Se ha realizado el ajuste del PID asociado al ángulo pitch siguiendo el procedimiento explicado en el apartado anterior. El punto deseado sobre el que se debería estabilizar el sistema se fija en cero grados. Ahora, comenzamos con una ganancia $K_p = 0{,}1$ cuya respuesta tiene una sobreoscilación de $\pm 10°$ y vamos incrementando su valor con incrementos de 0,5 unidades. La sobreoscilación aumenta a medida que aumenta la ganancia, con un valor de $K_p = 0{,}225$ el sistema oscila con un error de $\pm 15°$, pero el sistema aún no se ha vuelto inestable del todo. Eso ocurre cuando alcanzamos la ganancia límite de $K_p = 0{,}3$.

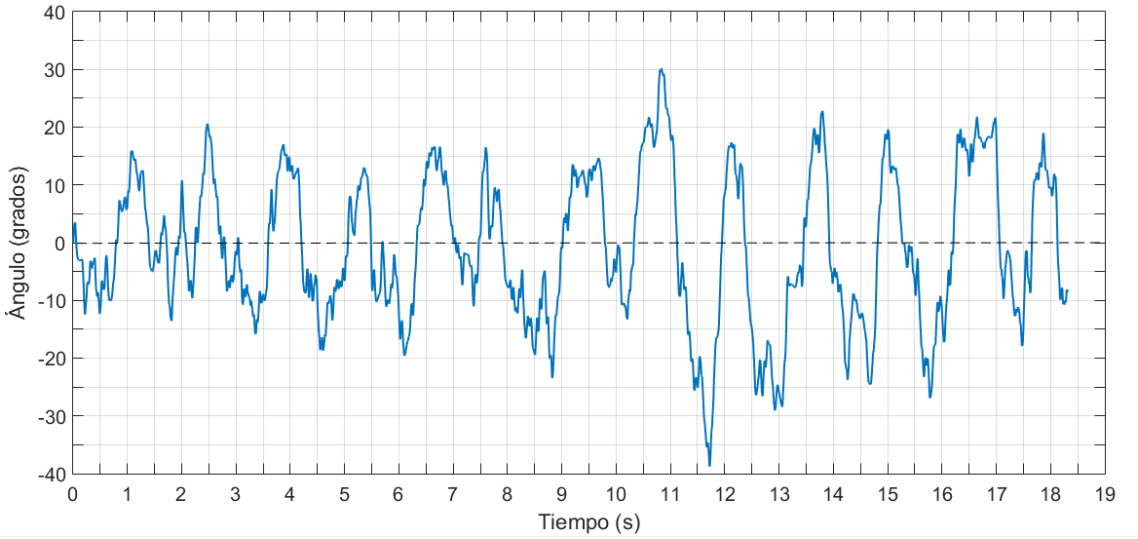

Figura 7.2 Respuesta real Pitch: Kp=0.4 (setpoint de 0°)

Como podemos observar el sistema oscila entre 30° y -40° y no consigue estabilizarse en torno al punto deseado (0 grados) por lo que se ha vuelto totalmente inestable. Por tanto, volvemos al último valor que daba una respuesta estable: $K_p = 0{,}225$.

Ahora, introducimos el término derivativo para disminuir la sobreoscilación y para que el sistema sea capaz de compensar el efecto de una perturbación externa. Éste término es difícil de ajustar y pequeñas variaciones producen grandes cambios en la respuesta del sistema. Arrancamos con un valor de partida de $K_p = 0{,}225$ y $K_d = 0{,}006$ obteniendo una respuesta que oscila en torno a $\pm 10°$. Aún no se ha reducido lo suficiente, por lo que aumentamos la $K_d = 0{,}011$, lo que provoca que el sistema se estabilice en 0 grados con una precisión de $\pm 5°$ y las oscilaciones a alta frecuencia son prácticamente inexistentes, es decir, tenemos una respuesta bastante mejor.

Incrementamos la ganancia para ver en qué punto aparecen oscilaciones a alta frecuencia, lo que ocurre con $K_d = 0{,}0195$. Con este último valor no se mejora la precisión de la respuesta por lo volvemos al anterior caso donde teníamos $K_p = 0{,}225$ y $K_d = 0{,}011$.

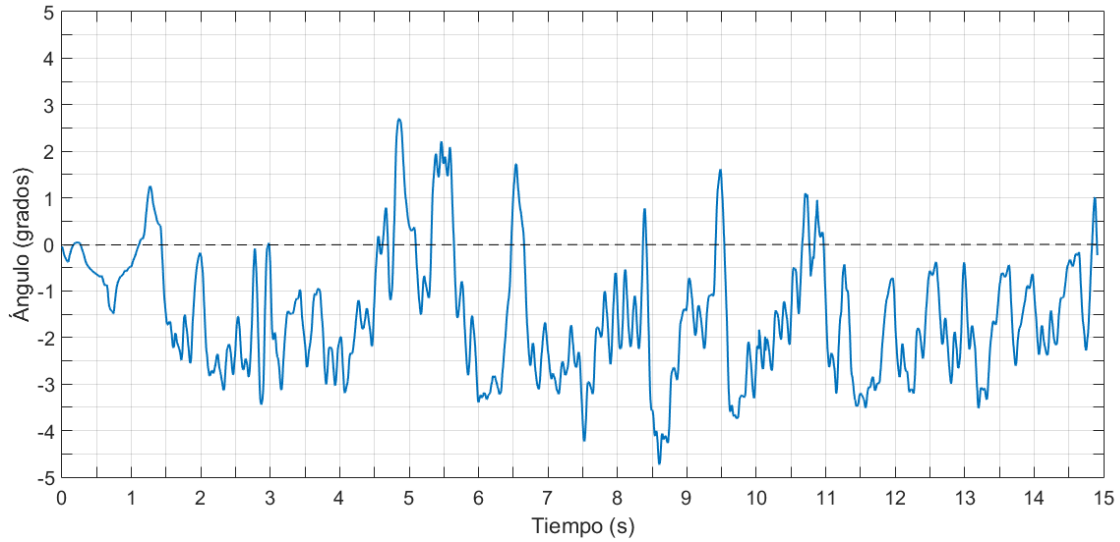

Figura 7.3 Respuesta real Pitch: Kp=0.4 Kd=0.011 (setpoint de 0°)

El regulador $K_p = 0{,}225$ y $K_d = 0{,}011$, estabiliza el sistema con un error de ±5°, sin embargo tiene error en régimen estacionario, por lo que no será capaz de alcanzar el punto deseado para todo el rango de trabajo UAV [-30°, 30°]. Razón por la cual incluimos el término integral.

La ganancia K_i se ajusta empezando por un valor de 0,02 con el que seguimos teniendo error en régimen estacionario, por lo que la aumentamos hasta $K_i = 0{,}07$ con el que obtenemos la mejor respuesta hasta el momento.

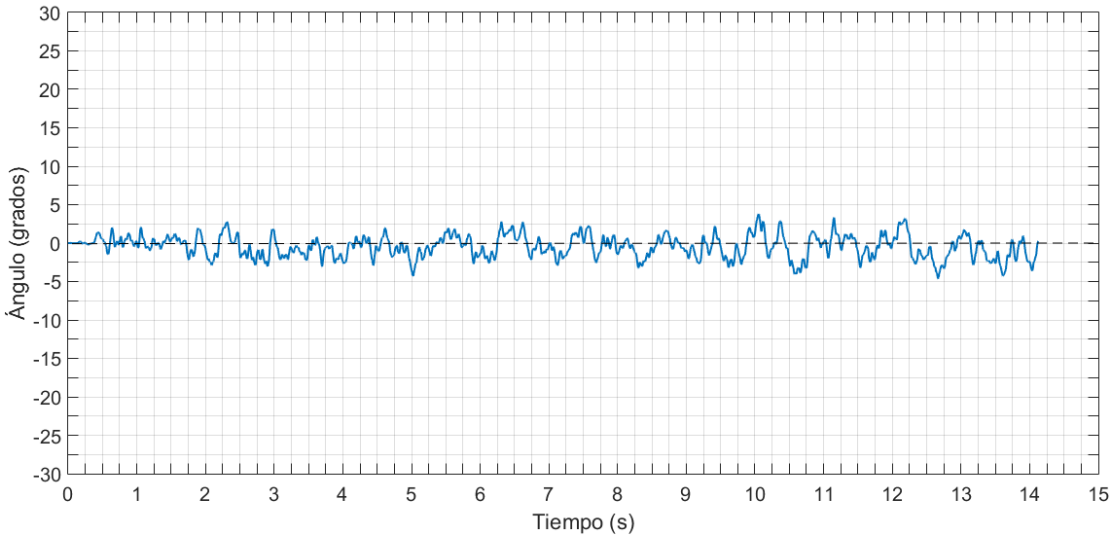

Figura 7.4 Respuesta real Pitch: Kp=0.225 Ki=0.07 Kd=0.011 (setpoint de 0°)

Como podemos observar, la respuesta obtenida con las constantes del regulador PID $K_p = 0{,}225$, $K_i = 0{,}07$ y $K_d = 0{,}011$ se estabiliza en torno al punto deseado con un error de ±5°, es decir ligeramente superior al esperado con las simulaciones en Matlab, y se ha reducido notablemente el error en régimen estacionario.

Ahora vamos a comprobar el comportamiento del regulador para un setpoint de 30 grados.

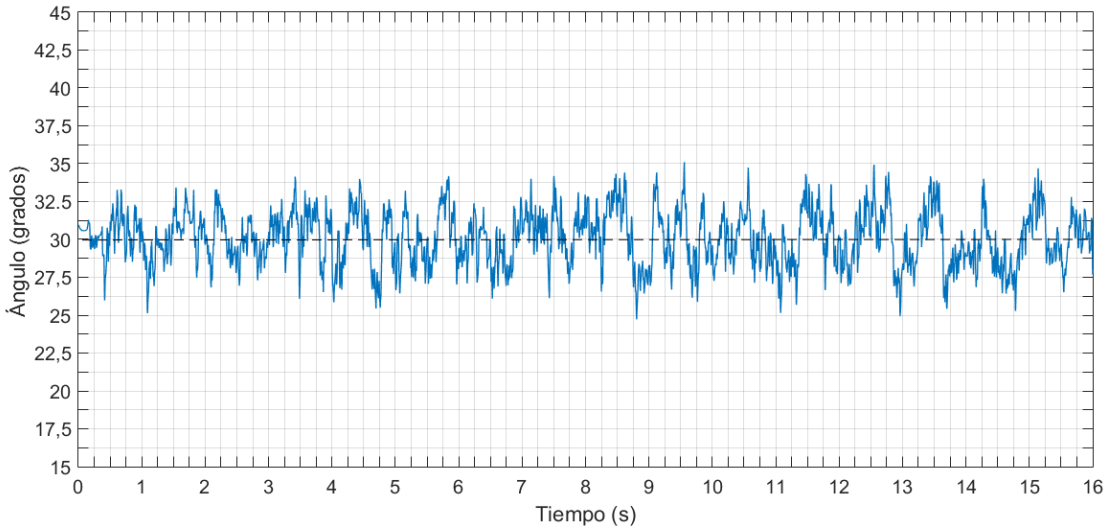

Figura 7.5 Respuesta real Pitch: Kp=0.225 Ki=0.07 Kd=0.011 (setpoint de 30°)

En la figura superior podemos observar que el regulador PID consigue alcanzar el punto deseado de 30° obteniendo una respuesta similar a la de la "Figura 7.4", por lo que no habrá problema en alcanzar cualquier setpoint que se encuentre en el rango de operación del UAV. En definitiva, el regulador que mejor realiza la estabilización del sistema, para todo el rango de operación y con el menor error posible es el PID cuyas constantes de control son $K_p = 0{,}225$, $K_i = 0{,}07$ y $K_d = 0{,}011$.

Por último, cabe destacar que al ser un cuadricóptero simétrico, los reguladores PID de los ángulos pitch y roll son exactamente iguales, por lo que sólo es necesario ajustar uno de ellos. En cuanto al regulador asociado al ángulo yaw, no se ha podido ajustar debido a la imposibilidad de sujetar de forma segura el UAV al banco de pruebas restringiendo así los movimientos de pitch y roll. Éste regulador se ajustará durante las pruebas de vuelo en el exterior, lo que no será un problema puesto que el ángulo yaw es el más facil de ajustar y no afecta a la estabilización del UAV.

Capítulo 8 Conclusiones y trabajos futuros

8.1 Conclusiones

Durante todo el proyecto se ha tratado de explicar el proceso de diseño y control de un cuadricóptero. Aunque el proyecto se centra en el sistema de control de vuelo del UAV, hemos decidido abordar todos los temas relacionados con el mismo. En primer lugar se han elegido los elementos que componen el quadcopter, cada uno de los cuales ha de ser seleccionado teniendo en cuenta las especificaciones del resto.

Después se han realizado los algoritmos que recogen la información de los sensores, comparando los valores obtenidos utilizando distintos métodos. A continuación se ha diseñado el driver de los motores, que nos permite enviar señales de control a los mismos, realizándose así los movimientos deseados. También se han implementado algoritmos de control PID que nos permiten ajustar el error cometido. En la "Figura 5.2" se encuentra un flujograma explicativo del funcionamiento conjunto del sistema de adquisición de datos y del sistema de control. Para facilitar el conexionado de los elementos, se ha diseñado una PCB, evitando así utilizar los cables que no son más que una carga.

Con el sistema completo diseñado, se ha empleado el modelado del UAV en la herramienta Simulink, para realizar simulaciones del sistema de control y hacernos una idea del valor que tendrán las constantes de control en el sistema real. Debido a que no se ha podido realizar las pruebas físicas para calcular los coeficientes de empuje y par del motor, el modelo en Matlab no es totalmente representativo del sistema real, por lo que las constantes de control ajustadas manualmente se han visto modificadas ligeramente con respecto a las obtenidas en simulación. Las pruebas del sistema de estabilización se han realizado limitando los grados de libertad a uno, el ángulo sobre el que se realiza el control.

Tras realizar los ensayos del sistema real en el banco de pruebas ilustrado en la "Figura 7.1", hemos hallado las ganancias del PID que consiguen estabilizar el UAV sobre los ángulos deseados. Con el regulador compuesto por $K_p = 0{,}225$, $K_i = 0{,}07$ y $K_d = 0{,}011$, se ha obtenido una respuesta adecuada que se estabiliza en torno al setpoint con un error de ± 5 grados y prácticamente sin error en el régimen estacionario. Además el regulador es capaz de alcanzar todo el rango de trabajo del UAV, como podemos observar en "Figura 7.4" y "Figura 7.5".

El regulador PID real es ligeramente menos preciso que el de las simulaciones en Matlab de la "Figura 6.9", lo cual es lógico puesto que en las simulaciones se suelen despreciar algunos efectos del sistema real. Aunque el regulador PID calculado se puede mejorar, hemos diseñado un sistema de control de vuelo bastante bueno que estabiliza el UAV correctamente.

Como el quadcopter utilizado es simétrico, las constantes de control valen para los reguladores de los ángulos pitch y roll. El ángulo yaw no precisa regulación en el banco de pruebas puesto que no afecta a la estabilización del sistema. Aunque no se ha podido realizar ninguna prueba de vuelo en el exterior, a la vista de los resultados obtenidos, no debería haber ningún problema en un vuelo real.

En cuanto al coste de los elementos embarcados en el cuadricóptero, que es posible consultar en el presupuesto incluido en el anexo "9.1", podemos afirmar que se ha diseñado un UAV de bajo coste y altas prestaciones.

Habida cuenta de los objetivos marcados y del trabajo desarrollado, podemos afirmar que se han alcanzado la mayoría de ellos obteniendo unos resultados satisfactorios, a excepción del objetivo de realizar las pruebas de vuelo en el exterior.

8.2 Trabajos futuros

A continuación se van a explicar las principales líneas de actuación que en mi opinión deberían seguirse para mejorar el sistema desarrollado en este proyecto. Estas mejoras hacen referencia tanto a mejorar el sistema de control diseñado como a la transformación del quadcopter en un UAV autónomo.

Tanto en los UAV autónomos como en los controlados remotamente, es fundamental la estimación del estado, que comprende la posición y orientación del cuadricóptero. En cuanto a la orientación, se propone complementar la información obtenida de la IMU con sensores de velocidad como el Optical Flow. Por otro lado, se podría utilizar una cámara para estimar la posición mediante el procesamiento de imágenes. Para fusionar los datos obtenidos de los distintos sensores, se propone utilizar el filtro de Kalman y comparar así los resultados con los obtenidos en este proyecto.

Con el fin de tomar conciencia del entorno que rodea al UAV se propone utilizar un sensor LIDAR de 360° que nos permitirá realizar un mapeo del lugar, lo que nos facilitará la labor cuando diseñemos el sistema de detección de obstáculos.

El tratamiento de imágenes y el mapeo del entorno precisan de una capacidad de procesamiento importante, por lo que habrá que realizar estas tareas en el procesador ARM Cortex-A9 de cuatro núcleos que contiene la placa UDOO.

En cuanto a modelado del UAV, se propone implementar un banco de pruebas para calcular experimentalmente los coeficientes de empuje y par del motor. De esta forma, la caracterización del modelo será más exacta y por consiguiente más representativo.

También se propone aumentar la frecuencia de trabajo de los ESC para analizar la influencia de ésta en el sistema de control. Hay que tener en cuenta que no todos los variadores electrónicos permiten aumentar la frecuencia de trabajo por encima de 50 Hz.

Finalmente, se propone medir la tensión de la batería LiPo de modo que a medida que ésta vaya disminuyendo, se modifiquen las constantes de control del PID. Esto compensaría el efecto que provoca la caída de tensión de alimentación en el control de vuelo del cuadricóptero.

En definitiva, aplicando estas mejoras al sistema de control desarrollado en este proyecto, se logrará un sistema de control bastante robusto, con un conjunto amplio de sensores que nos permitirán dotar al UAV de un grado de autonomía considerable.

Capítulo 9 Anexos

9.1 Presupuesto

Código	Unidad	Descripción	Medición	Precio unitario	Precio total
1		**Capítulo 1: Hardware**			
1 .01	ud.	**Placa UDOO Quad** Suministro de placa UDOO Quad, CPU Freescale i.MX 6 ARM Cortex-A9 Quad core 1GHz, Atmel SAM3X8E ARM Cortex-M3 CPU (Arduino Due), GPU Vivante GC 2000 + Vivante GC 355 + Vivante GC 320, RAM DDR3 1GB, modulo WiFi.	1	120,60 €	120,60 €
1 .02	ud.	**Starter Kit para UDOO Quad** Suministro de Starter Kit que contiene cable HDMI, cargador EU, adaptador USB, cable USB a Micro USB, cable de alimentación SATA, micro SD de 8 Gb y batería externa para el RTC.	1	26,80 €	26,80 €
1 .03	ud.	**Motor Brushless Turnigy L2210C** Suministro de Turnigy L2210C Brushless Motor, 150 W, 1200 Kv, 15,8 A corriente max, peso total de 63g, voltaje 7.2V-11.1V, 700g de empuje aproximadamente.	4	11,50 €	46,00 €
1 .04	ud.	**Pareja de helices ABS 9X4,7** Suministro de pareja de helices 9x4.7, una normal y otra contrarrotatoria, material ABS y adaptadores para distintos diametros de eje.	2	3,55 €	7,10 €
1 .05	m.	**ESC Turnigy Multistar 20A Slim BLHeli** Suministro de Turnigy Multistar Slim BLHeli, 20A, alimentación con batería LiPo de 2-6 celdas, BEC, conectores de 3.5mm, 20.3g de peso.	4	9,54 €	38,16 €
1 .06	ud.	**Transmisor y receptor RC Turnigy 9X** Suministro de pack Turnigy 9x, incluye transmisor Turnigy 9x 9Ch, modulo Turnigy RF9X-V2 y receptor Turnigy 9X8C-V2. Display de 128*64 LCD, soporta tipos Heli/Acro/Glid.	1	53,39 €	53,39 €
1 .07	ud.	**Adafruit 10 DOF IMU Breakout** Suministro de Adafruit 10-DOF IMU Breakout. Contiene L3GD20 + LSM303 + BMP180, peso de 2.8g, comunicación I2C, dimensiones de 23x38x3 mm	1	26,75 €	26,75 €

Código	Unidad	Descripción	Medición	Precio unitario	Precio total
1 .08	ud.	**Batería RCInnovations Desire power**			
		Suministro de batería RCInnovations Desire power 6000 mah, 3 celdas, 30C, 420g de peso.	1	55,50 €	55,50 €
1 .09	ud.	**Batería parrot ARdrone 2.0**			
		Suministro de Batería parrot ARdrone 2.0, 1500 mah de capacidad, corriente máxima de 1.5 A y peso de 120g.	1	19,00 €	19,00 €
1 .10	ud.	**HK Quadcopter Power distribution board**			
		Suministro de HK Quadcopter Power distribution board, conector de alimentación de entrada XT60, conectores de salida 4 x 3.5mm Female bullet, peso de 27.3g.	1	3,55 €	3,55 €
		SUBTOTAL Capítulo 1: Hardware			**396,85 €**
2		**Capítulo 2: Software**			
2 .01	ud.	**Arduino IDE**			
		Licencia de uso Arduino IDE.	1	0,00 €	0,00 €
2 .02	ud.	**Matlab Student Suite**			
		Licencia de uso que incluye Matlab básico, Simulink, Control System Toolbox, Simulink Control Design, Image Processing Toolbox, Optimization Toolbox, Signal Processing Toolbox, entre otros.	1	69,00 €	69,00 €
		SUBTOTAL Capítulo 2: Software			**69,00 €**
3		**Capítulo 3: Recursos humanos**			
3 .01	h	**Ingeniero electrónico industrial**			
		Coste del trabajo realizado como ingeniero electrónico industrial. Expresado en €/h.	300	18,750 €	5.625,00 €
		SUBTOTAL Capítulo 3: Recursos humanos			**5.625,00 €**

RESUMEN DE PARTIDAS	
SUBTOTAL Capítulo 1: Hardware	**396,85 €**
SUBTOTAL Capítulo 2: Software	**69,00 €**
SUBTOTAL Capítulo 3: Recursos humanos	**5.625,00 €**
TOTAL PROYECTO	**6.090,85 €**

9.2 Diagrama de tiempos del proyecto

	TAREAS	ENERO				FEBRERO				MARZO				ABRIL				MAYO				JUNIO				JULIO				AGOSTO				SEPTIEMBRE			
		S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
1	Estado del arte																																				
2	Selección de los elementos																																				
3	Placa UDOO. Funcionamiento																																				
4	Montaje del quadcopter																																				
5	Prueba de los elementos																																				
6	Lectura de datos del receptor RC																																				
7	Lectura de datos de la IMU																																				
8	Driver de los motores																																				
9	Diseño del sistema de control																																				
10	Modelado del UAV																																				
11	Ensayos realizados. Ajuste del PID																																				
12	Memoria																																				

Tabla 9.1 Diagrama de tiempos del proyecto

Bibliografía

[1] H. Voos, «Nonlinear control of a quadrotor micro-UAV using feedback-linearization», en Mechatronics, 2009. ICM 2009. IEEE International Conference on, 2009, pp. 1–6.

[2] S. Bouabdallah, P. Murrieri, y R. Siegwart, «Towards autonomous indoor micro VTOL», Autonomous robots, vol. 18, n.º 2, pp. 171–183, 2005.

[3] P. Pounds, R. Mahony, y P. Corke, «Modelling and control of a quad-rotor robot», en Proceedings Australasian Conference on Robotics and Automation 2006, 2006.

[4] T. Luukkonen, «Modelling and control of quadcopter», Independent research project in applied mathematics, Espoo, 2011.

[5] M. Euston, P. Coote, R. Mahony, J. Kim, y T. Hamel, «A complementary filter for attitude estimation of a fixed-wing UAV», en 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008, pp. 340–345.

[6] G. Mao, S. Drake, y B. D. Anderson, «Design of an extended kalman filter for uav localization», en Information, Decision and Control, 2007. IDC'07, 2007, pp. 224–229.

[7] L. Di, T. Fromm, y. Chen, «A data fusion system for attitude estimation of low-cost miniature UAVs», Journal of Intelligent & Robotic Systems, vol. 65, n.º 1-4, pp. 621–635, 2012.

[8] P. Martin y E. Salaun, «The true role of accelerometer feedback in quadrotor control», 2010.

[9] J. L. Crassidis, F. L. Markley, y. Cheng, «Survey of nonlinear attitude estimation methods», Journal of guidance, control, and dynamics, vol. 30, n.º 1, pp. 12–28, 2007.

[10] L. Sevilla Fernández, «Modelado y control de un cuadricóptero».

[11] S. Bouabdallah, A. Noth, y R. Siegwart, «PID vs LQ control techniques applied to an indoor micro quadrotor», en Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, 2004, vol. 3, pp. 2451–2456.

[12] E. Reyes-Valeria, R. Enriquez-Caldera, S. Camacho-Lara, y J. Guichard, «LQR control for a quadrotor using unit quaternions: Modeling and simulation», en Electronics, Communications and Computing (CONIELECOMP), 2013 International Conference on, 2013, pp. 172–178.

[13] K. Turkoglu, U. Ozdemir, M. Nikbay, y E. M. Jafarov, «PID parameter optimization of an UAV longitudinal flight control system», International Journal of Mechanical, Industrial Science and Engineering, vol. 2, n.º 9, 2008.

[14] B. Kada y. Ghazzawi, «Robust PID controller design for an UAV flight control system», en Proceedings of the World Congress on Engineering and Computer Science, 2011, vol. 2, pp. 19–21.

[15] M. R. R. Khoygani, S. Hajighasemi, y D. Sanaei, «Designing and simulation for vertical moving control of UAV system using PID, LQR and Fuzzy Logic», International Journal of Electrical and Computer Engineering, vol. 3, n.º 5, p. 651, 2013.

[16] M. Santos, V. Lopez, y F. Morata, «Intelligent fuzzy controller of a quadrotor», en Intelligent Systems and Knowledge Engineering (ISKE), 2010 International Conference on, 2010, pp. 141–146.

[17] G. V. Raffo, M. G. Ortega, y F. R. Rubio, «An integral predictive/nonlinear H∞ control structure for a quadrotor helicopter», Automatica, vol. 46, n.º 1, pp. 29–39, 2010.

[18] J. López, R. Dormido, S. Dormido, y J. P. Gómez, «A Robust Controller for an UAV Flight Control System», The Scientific World Journal, vol. 2015, 2015.

[19] J. Dunfied, M. Tarbouchi, y G. Labonte, «Neural network based control of a four rotor helicopter», en Industrial Technology, 2004. IEEE ICIT'04. 2004 IEEE International Conference on, 2004, vol. 3, pp. 1543–1548.

[20] T. Dierks y S. Jagannathan, «Neural network output feedback control of a quadrotor UAV», en Decision and Control, 2008. CDC 2008. 47th IEEE Conference on, 2008, pp. 3633–3639.

[21] «Mini Tricopter thoughts – need feedback – RCExplorer». [En línea]. Disponible en: http://rcexplorer.se/blog/2015/08/mini-tricopter-thoughts-need-feedback/. [Accedido: 29-ago-2016].

[22] Thingiverse.com, «ExPrezes 3D printed Spyda 500 Quadcopter by Gyrobot». [En línea]. Disponible en: http://www.thingiverse.com/make:131541. [Accedido: 29-ago-2016].

[23]«Quad configuration — Copter documentation». [En línea]. Disponible en: http://ardupilot.org/copter/docs/connect-escs-and-motors.html. [Accedido: 29-ago-2016].

[24]«Extreme RC World. Walkera Black Tali H500 Hexacopter RTF». [En línea]. Disponible en: https://www.xrcworld.com/walkera-black-tali-h500-hexacopter-rtf-iuas. [Accedido: 29-ago-2016].

[25] L. C. B. H. Valdés-González, «Leak detection in water pipelines: Proposal based on a bank of filters», Rev Chilena de Ingenieria, vol. 17, n.º 3, pp. 375–385, 2009.

[26]«UDOO QUAD – SHOP – UDOO». [En línea]. Disponible en: http://shop.udoo.org/eu/quad-dual/udoo-quad.html. [Accedido: 29-ago-2016].

[27]«Electric Drives - Brushless DC and Reluctance Motors - Description and Applications». [En línea]. Disponible en: http://www.mpoweruk.com/motorsbrushless.htm. [Accedido: 29-ago-2016].

[28]«Helices multicoptero ABS 9x4.7 (pareja)», RC Innovations. [En línea]. Disponible en: https://rc-innovations.es/multicopter-props-9x4.5. [Accedido: 29-ago-2016].

[29]«Desire power V8 Series 3s 6000mAh 30C», RC Innovations. [En línea]. Disponible en: https://rc-innovations.es/desire-power-3s-6000-30c. [Accedido: 29-ago-2016].

[30]«Turnigy 9X 9Ch Transmitter w/ Module & 8ch Receiver (Mode 2) (v2 Firmware)», HobbyKing Store. [En línea]. Disponible en: http://www.hobbyking.com/hobbyking/store/uh_viewitem.asp?idproduct=8992. [Accedido: 29-ago-2016].

[31]«Ejes de referencia de UAV». [En línea]. Disponible en: http://www.prometec.net/wp-content/uploads/2015/10/PRY.gif. [Accedido: 29-ago-2016].

[32]«Arduino Playground - Appendix3». [En línea]. Disponible en: http://playground.arduino.cc/ArduinoNotebookTraduccion/Appendix3. [Accedido: 30-ago-2016].

[33]«Señal PPM - AEROMODELISMO VIRTUAL». [En línea]. Disponible en: http://www.aeromodelismovirtual.com/showthread.php?t=34086. [Accedido: 30-ago-2016].

[34]«Adafruit 10-DOF IMU Breakout - L3GD20H + LSM303 + BMP180 ID: 1604 - $29.95 : Adafruit Industries, Unique & fun DIY electronics and kits». [En línea]. Disponible en: https://www.adafruit.com/product/1604. [Accedido: 30-ago-2016].

[35]«Arduino UNO Rev3», Arduino Store USA. [En línea]. Disponible en: http://store-usa.arduino.cc/products/a000066. [Accedido: 30-ago-2016].

[36]«dch33/Quad-Sim», GitHub. [En línea]. Disponible en: https://github.com/dch33/Quad-Sim. [Accedido: 30-ago-2016].

[37]«Turnigy L2210C-1200Kv». [En línea]. Disponible en: http://bhabbott.net.nz/L2210C-1200.html. [Accedido: 30-ago-2016].

[38] B. Beauregard, «Improving the Beginner's PID», Web. http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-direction, 2012.

[39] «ar.drone, drone, quadricopter, parrot, wifi, spare parts». [En línea]. Disponible en: http://www.store-parrot.com.au/hd-battery-high-density-1500-mah-for-ar-drone-2-0-range.html. [Accedido: 12-sep-2016].